DOCUMENT RESUME

ED 169 912 **IR 007 197**

TITLE	Computer Manual.
INSTITUTION	Illinois State Office of Education, Springfield.
PUB DATE	78
NOTE	84p.; Not available in hard copy due to print quality of original document
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	Computers; *Computer Science; Flow Charts; Guides; *Instructional Aids; Instructional Materials; Linear Programing; *Programing Languages

ABSTRACT

This manual designed to provide the teacher with methods of understanding the computer and its potential in the classroom includes four units with exercises and an answer sheet. Unit 1 covers computer fundamentals, the mini computer, programming languages, an introduction to BASIC, and control instructions. Variable names and constants described in Unit 2 include the LET instruction, GO TO statement, IF--THEN statement, and INPUT instruction. READ and DATA statements, summing and counting, functions, and the FOR and NEXT statements are covered in Unit 3, while Unit 4 includes defining a function--DEF, subscripted variables, subroutines--GOSUB, and graphing--TAB function. Control commands, error messages, editing, and flow charting are described in the appendices, as well as the "mailbox analogy," which illustrates the operation of the digital computer. (JEG)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED169912

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

COMPUTER MANUAL

BEST COPY AVAILABLE

State Board of Education

Illinois Office of Education

Joseph M. Cronin

State Superintendent

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

C. Reisinger

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC) AND USERS OF THE ERIC SYSTEM."

Revised 1974

Reprinted 1978

IR007197

TABLE OF CONTENTS

	Page
UNIT I	1-10
Computer Fundamentals	1
The Mini Computer	3
Languages	4
Introduction to BASIC	4
BASIC Statements-PRINT	5
Control Instructions	9
UNIT II	11-27
Variable Names and Constants	11
LET Instruction	12
GO TO Statement	16
IF – THEN Statement	18
INPUT Instruction	22
UNIT III	28-45
READ and DATA Statements	28
Summing and Counting	30
Functions	35
The FOR and NEXT Statements	40
UNIT IV	46-58
Defining a Function – DEF	46
Subscripted Variables	48
Subroutines – GOSUB	53
Graphing – TAB Function	55
APPENDICES I, II, and III	59-78
I - Control Commands, Error Messages, and Editing	59
II - Flow Charting	65
III - A Useful Analogy	70
ANSWERS to EXERCISES	79

FOREWORD

The computer has made a tremendous impact on modern society, transforming business, industry, transportation, recreation, education, and virtually all other aspects of the lives of our citizens.

Today's students are truly children of the computer age, and we, as educators, must provide them with the opportunity to cope with this incredible machine--its power, its potential and its limitations.

The responsibility for computer education is shared by many curriculum areas. As our educational system accepts this responsibility, teachers must be provided with methods of understanding the nature of the computer and the potential of the computer in the classroom. This manual has been produced to assist the teacher in these endeavors.

The Illinois Office of Education is indebted to Mr. John G. Stoudt and Mr. Walter Clevenger for the preparation of the original version of Computer Manual.

For additional copies of this publication, contact the Program Services Team member assigned to your district or write Program Planning and Development, Illinois Office of Education, 100 North First Street, Springfield, Illinois 62777.

Joseph M. Cronin
State Superintendent of Education
Illinois Office of Education

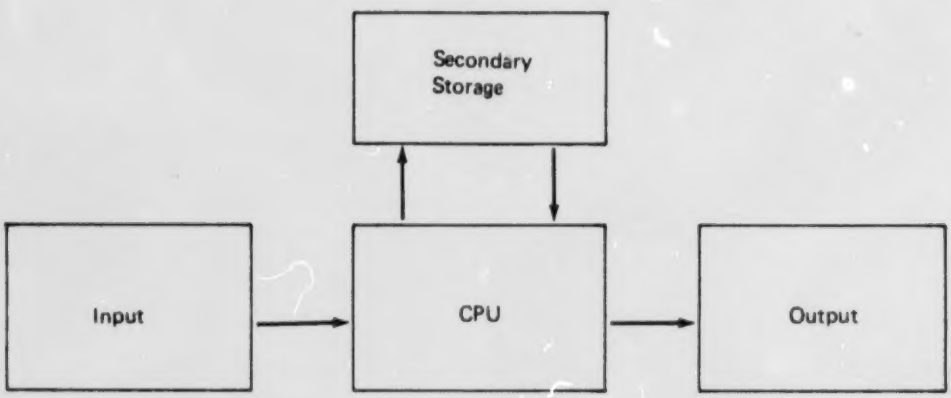
UNIT I
(Pages 1-8)

UNIT I

COMPUTER FUNDAMENTALS

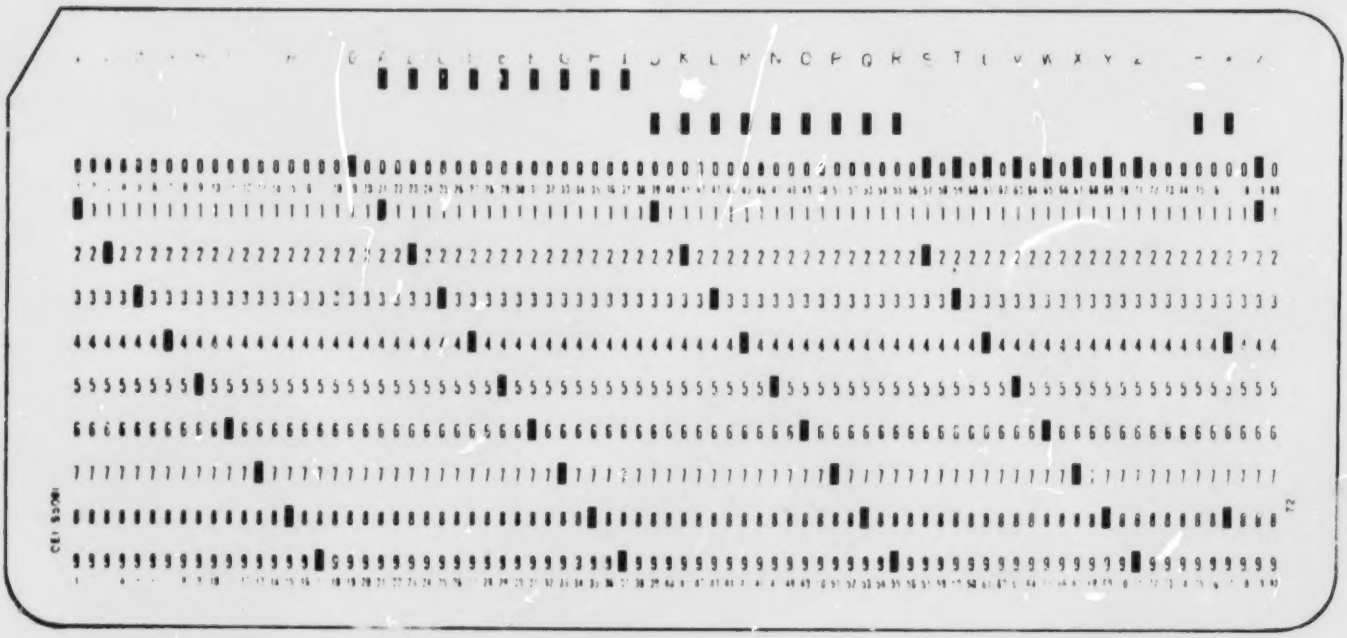
Although the intent of this manual is not to teach all the intricacies of the digital computer, it will be helpful to look at certain aspects of its internal operation to better understand the use of a programming language.

There are four functional aspects of a computer system: 1) input, 2) output, 3) central processing unit, and 4) secondary storage.



Input Devices

In order to get information into the computer, an **input** device is needed. There are a number of input devices in use, such as the punched card reader, the teletypewriter, the optical character reader (OCR), and the magnetic ink character recognition (MICR). The most common input device at present is the punched card reader, a device which "reads" the arrangement of punched holes in a punched card. The card is "read" by photoelectric light or brush. Card readers may operate with speeds of 100 to 2000 cards a minute.



Output Devices

Information manipulated by the computer may be brought out of the system by means of an output device. Output may be obtained through a teletypewriter, a card punch, a paper tape punch, or, the most common device, the line printer.

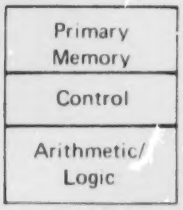
Most printers are impact printers. These print by pressing the paper and ribbon against a raised symbol. Impact printers may print from 300 to 3000 lines a minute.

The non-impact printer may print more than 3000 lines a minute. In this device an electrostatic charge is placed on the paper and the paper is dusted with ink particles.

The printers of the future may produce microfilm. With the immense amount of paper being belched out of printers today, it may not be long before we will be buried in paper. These printers may print from 7000 to 30,000 lines a minute and take up less space than conventional printers. A report of 1500 pages might be held in the palm of the hand.

Central Processing Unit

The central processing unit or CPU is the computer. Other devices in a computer system are called peripheral equipment. The CPU is made up of three parts: 1) primary memory, 2) arithmetic and logic, and 3) control.



The CPU

The **primary memory** is where the **instructions** (program) and **data** are stored in preparation for the processing of the program. A memory of "4K" has 4000 locations or positions for storage.

The **arithmetic/logic** unit performs the fundamental arithmetic operations and logic operations.

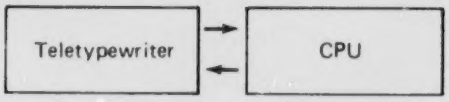
The **control unit** is that unit which directs data from input to memory - to arithmetic - to memory - to output, etc. It directs the execution of the program and coordinates the operations of the CPU.

Secondary Storage

Secondary storage might be considered an extension of the primary memory. Such storage may be performed by magnetic tape, magnetic drum, or magnetic disk drives. These units are brought into play to assist the relatively small memory of the CPU.

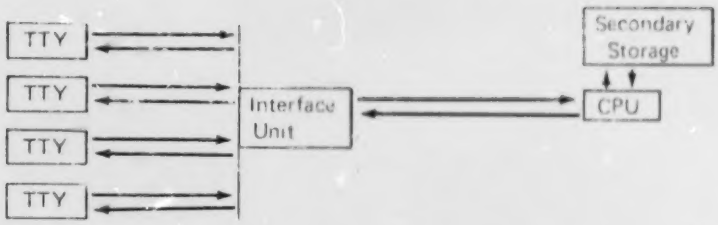
THE MINI-COMPUTER

Of interest to teachers of secondary school mathematics and science, as well as business and industry, is the mini-computer. We might stop at this point to describe the configuration of a typical mini-computer.



As can be seen, the teletypewriter acts as both an input and output (I/O) device. The CPU provides sufficient (perhaps 4-12K) memory for the problems encountered by the mini-computer.

Such a system might be expanded to a time-sharing configuration where a number of teletypewriters share the CPU. Additional storage may be necessary.



Time Sharing System

LANGUAGES

It would be virtually impossible to prepare a computer to "understand" our instructions to it if given in English, with all the complexity and variations of the language. Communication with the earliest computers was at another extreme as we were forced to "speak" in terms of binary numbers, the fundamental language of the computer.

But this advancing science has brought us a means of bridging the gap between our language and that of the computer. The compiler, translator, and interpreter are means of **converting** a "high-level language" such as FORTRAN, COBOL, ALGOL, or BASIC into a machine language that the computer itself can respond to.

The compiler and translator are programs which convert each "macro" instruction of the high-level language into a series of machine instructions.

The BASIC language is a powerful conversational mode language. It makes use of common English words and algebraic symbols which lend themselves to easy comprehension. BASIC, like FORTRAN and ALGOL, is oriented to scientific programming and is especially suited to time-sharing systems.

INTRODUCTION TO BASIC

The number of BASIC instructions that a given system has available is relative to the size of memory in the computer system and the flexibility of the software. The "software" is the compiler or translator program for BASIC, the library of programs, instructions, etc. The following instructions will be included in virtually all BASIC software packages.

PRINT	IF – THEN
READ	FOR – NEXT
DATA	DIM
INPUT	REM
LET	GO TO
GOSUB	END

Before these instructions are discussed, it may be wise to look at the fundamental arithmetic operations as they are used in BASIC.

Arithmetic Operations

Arithmetic Expression	BASIC Symbol	BASIC Expression
$4 + 2$	+ (addition)	4 + 2
$3 - 2$	- (subtraction)	3 - 2
$6 \div 5$	/ (division)	6 / 5
$3 \cdot 4$	* (multiplication)	3 * 4
3^2	↑ (exponentiation)	3 ↑ 2

As in algebraic systems, there is an **order** of operations observed in BASIC as the computer scans the expression from left to right.

The order of priority is as follows:

1) quantities inside parentheses are computed
2) exponentiation (from left to right)
3) multiplication and division (from left to right)
4) addition and subtraction (from left to right)

Examples:

3 + 6 ↑ 2 = 3 + 36 = 39
7 - 4 ↑ 3 / 2 = 7 - 64 / 2 = 7 - 32 = -25
5 ↑ (2 + 1) = 5 ↑ 3 = 125
18 / 3 * 2 = 6 * 2 = 12
18 / (3 * 2) = 18 / 6 = 3
5 + 3 / 2 = 5 + 1.5 = 6.5
15 - (6 * (3 + 2) - 20) + 3↑2 = 15 - (6 * 5 - 20) + 3↑2
= 15 - (30 - 20) + 3 ↑ 2
= 15 - 10 + 3 ↑ 2
= 15 - 10 + 9
= 5 + 9
= 14

Exercises:

Evaluate the following:

1. 12 / 4 * 3
2. 5 * 3 ↑ 2
3. 4 ↑ 1 + 2
4. (5 - 2) ↑ 3
5. 5 - 2 ↑ 3
6. 4 ↑ (1 + 2)
7. (2 ↑ (1 ↑ 4 * 0.5)) + 3
8. 5 / 2 * 4 + 2 * 3
9. 5 / 2 * (4 + 2) * 3
10. (5 * 2) * 3 - 4 / 8

BASIC STATEMENTS

Perhaps the best instruction to take up first is the word PRINT. The PRINT statement commands the computer to output various data.

PRINT Format

_____ PRINT _____
(statement number) (variables, arithmetic expressions, alphameric text)

Characteristics:

1. This instruction commands the computer to deliver data to an output device.

2. It will output the **values** of variables and expressions; it will output alphameric (ALPHAbetic-nuMERIC) text; or it will output combinations of these.

3. Alphameric text, enclosed in quotation marks, will be printed verbatim, including spaces. This kind of output can be used for column headings, etc.

4. In the expression following PRINT, commas will cause data to be printed in five columns across the page, semicolons can be used to print data in more compact form.

Note: A statement number must precede the PRINT instruction in a program. The role of these numbers will evolve in subsequent discussion.

The following examples will illustrate the use of PRINT in evaluating expressions and to output text. The PRINT instruction as it pertains to variables will be discussed later.

Examples:

```
 10 PRINT 3+4*6 <-------------- PROGRAM
 20 END
RUN <------------------------- Entered by Operator
 27 <-------------- OUTPUT
```

Observe: The END instruction signals the **end** of the program.

Observe: RUN is entered **via the terminal,** without a statement number, commanding the computer to begin processing the program.

```
 100 PRINT 6↑2,6+2,6-2,6/2,6*2
 200 END
RUN
 36            8             4             3             12
```

Observe: The commas in the PRINT statement cause the data to be printed in five columns, a standard BASIC format.

```
100 PRINT 6↑2;6+2;6-2;6/2;6*2
200 END
RUN
 36  8  4  3  12
READY
```

Observe: The semicolon causes output in more compact form.

11

```
   8 PRINT "MY NAME IS JOHN DOE"
   9 END
RUN
MY NAME IS JOHN DOE
```

Observe: The computer reproduces symbols betwecn quotation marks verbatim.

```
READY

70 PRINT "THE SUM OF 3 AND 4 IS"; 3+4
80 END
RUN
THE SUM OF 3 AND 4 IS  7

READY
```

Observe: The sum of the expression 3 + 4 is evaluated and printed **following** the text. In most BASIC software, the semicolon may be omitted and the same results obtained.

```
   10 PRINT "JOHN DOE"
   20 PRINT
   30 PRINT "ANYTOWN, U.S.A."
   40 END
RUN
JOHN DOE

ANYTOWN, U.S.A.
```

PROGRAM

Entered by Operator

OUTPUT

Observe: 20 PRINT is used to skip a line.

```
  10 PRINT "NOW IS THE TIME FOR";
  20 PRINT " ALL GOOD MEN TO ----"
  30 END
RUN
NOW IS THE TIME FOR ALL GOOD MEN TO ----

READY
```

Observe: If a semicolon or comma is the last character in a PRINT instruction, there will be no carriage return or line feed.

Exercises:

Express the appearance of the output derived from the following programs:

1.
```
10 PRINT 6+3/4, 0·6*2, 0·3↑3
20 PRINT 6+3/4; 0·6*2; 0·3↑3
30 END
```

2.
```
 5 PRINT "THE EXPRESSION 7*(8-2)/5 EQUALS"
10 PRINT 7*(8-2)/5
15 END
```

3. 100 PRINT "SUM", DIFFERENCE", "PRODUCT", "QUOTIENT", "EXPONENTIAL"
   110 PRINT .045+189, 67.41-5.37, 4.8*0.5, 56.0/.8, 49↑.5
   120 END

4. 10 PRINT "THE COORDINATES ("3+4-5 ","7.0*2-9 ") DESIGNATE";
   20 PRINT " THE POINT P1."
   30 END

The preceding examples are very simple "programs" complete with statement numbers and an END instruction.

END Format:

_____ END
(Statement number)

Characteristics:

1. This instruction tells the computer that it has received the last instruction on a given program.

2. This is a non-executable instruction. That is, in the execution of the program, the computer ignores this instruction.

There is another statement that might be discussed at this time. This is a non-executable statement used to insert explanatory remarks into the program. The program can therefore be identified with a name, the date, the name of the programmer, and other documentation.

REM Format:

_____ REM _____
(statement number) (any alphameric expression)

Characteristics:

1. This is a non-executable instruction, and can therefore be placed anywhere in the program before the END statement.

2. REM can be followed by any alphameric expression such as names, dates, explanations of the program, etc.

The following illustrate the REM statement:

```
  10 REM W. CLEVENGER 7/15/70
  20 REM VERY SIMPLE PROGRAM
  30 PRINT 4+5*2.1
  40 END
RUN
 14.5

READY
```

```
READY

10 REM J. STOUDT 7/16/70
20 REM JUST AS SIMPLE
30 PRINT "THIS IS A SIMPLE PROGRAM"
40 PRINT "THE 2ND , 3RD, AND 4TH POWERS OF 3 ARE";
50 REM NOW HERE COMES THE BIGGEE
70 PRINT 3↑2; 3↑3; 3↑4
80 END
RUN
THIS IS A SIMPLE PROGRAM
THE 2ND , 3RD, AND 4TH POWERS OF 3 ARE 9  27  81

READY
```

It will be wise practice **at least** to identify the programmer in all the programs that you write.

CONTROL INSTRUCTIONS

There are "control" instructions such as END in BASIC which require a statement number and as such are part of the program.

In order to run a program, there are other commands that must be given through the **input device.** These commands will vary from one software package to another. The following illustrates:

```
  10 PRINT 3+4
  20 END
RUN
 7

READY
```

These commands do not use statement numbers.

RUN, entered by the operator, indicates to the computer's software to branch back to the first executable instruction and begin the run of the program.

READY is **the computer's response** indicating that it is ready to accept a new program. We are now ready to begin writing and processing simple programs; we can explore the many possibilities of the versatile PRINT instruction in BASIC.

See Appendix I for additional discussion of these control statements.

In all the programs thus far, conventional spacing has been used between words and numbers for the **purpose of clarity. BASIC ignores spaces** except as desired in printed alphameric text.

Exercises:

1. Write a program to print your name, address, and zip code, just as it might appear on a letter envelope.

2. Write a program to print the sum, difference, product, and quotient (in four columns) of the following number pairs. Use a line for each number pair.

   5, 6 7.1, -3 -0.003, 879 1.4, 9.2

3. Write a program which combines text and an evaluated expression on the same line.

4. Rewrite the program of exercise No. 2, adding column headings like "Sum," "Difference," etc.

5. Rewrite the program of exercise No. 4, placing the number pair itself in the first column with an appropriate heading.

6. Write a program using both the comma and semicolon in PRINT statements so as to explore their effect.

NOTE: If you have progressed to the point where the computer has issued you an error message, refer to Appendix I.

NOTE: See page 67 for a picture of the teletypewriter keyboard.

UNIT II

VARIABLE NAMES AND CONSTANTS

In the BASIC language suitable variable names for numbers may take any of three forms. A variable may be indicated by

1. A single letter of the alphabet - or
2. A single letter of the alphabet followed by a single digit 0-9 - or
3. A subscripted variable (more on this one later).

Below are some acceptable variable names.

A	Z	W7	
A1	M	C	
M9	L2	B(1)	(subscripted variable)
X	B	Y(3, 4)	(subscripted variable)

It is a good practice to choose a variable name which is easily identified with the data it represents, like A for area and F for frequency.

Entering Constants

Constants can be **entered** in a program in either of three modes:

1. fixed point, like 613 -7, 14, -2013
2. floating point, like 23.07, -0.4516, .6
3. exponential, like 6.13 E+02, 13 E-6

Note: E means "times 10 to the power of."

The following examples should better illustrate the exponential forms which are acceptable:

Algebraic expression	Acceptable BASIC forms
2.3×10^4	2.3E04, 2.3E4, 2.3E+4, .23E5, 23E3, etc.
-12.06×10^{-9}	-12.06E-9, -12.06E-09, -1.206E-8, -120.6E-10, etc.

If neither a plus or minus sign is included, a positive exponent is assumed.

Output of Constants

BASIC will output numerical data according to the following:

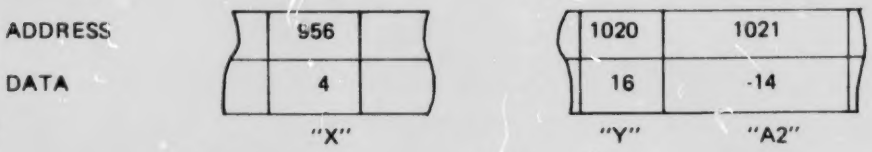
1. integer data will be printed in fixed point format
2. non-integer data will be expressed in floating point format
3. numbers very large or very small will automatically be output in exponential form

A WORD ABOUT MEMORY

Variable names, as far as a computer is concerned, should be thought of as **names for locations in memory.**

For example, when we "LET x = 4" we are storing the data 4 in a location in memory called x.

Internally, the computer refers to this location or position in memory by its address. We are seldom concerned what this address might be.



In the above illustrations, the variable

X represents the location (address) 956,

Y represents the location 1020, and

A2 represents 1021

LET statements enable us to change the data in these locations.

```
SCR

READY

  10 LET X=4
  20 LET X=6
  30 LET X=10
  40 PRINT X
  50 END
RUN
 10

READY
```

Observe: The location in memory called X first holds a data value of 4, then 6, then 10. When the control encounters statement 40, the current value (10) of X is printed. The values of 4 and 6 are destroyed when subsequent assignments were made to X.

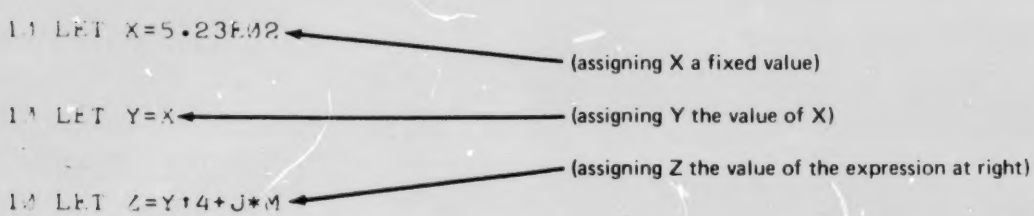
In summary then, the LET instruction is used to assign a value to a variable name. The value may be obtained from a single number, another variable, or an arithmetic expression.

LET Format:

_____ LET _____ = _____

(statement number) (variable name) (single number, variable name, or arithmetic expression)

To illustrate the LET format:



Below is a program which illustrates the use of a variable name in the LET and PRINT statements.

```
READY

  10 LET X=7+3
  20 PRINT X
  30 END
RUN
 10

READY
```

Other examples of the LET and PRINT statements can be seen in the programs which follow:

```
BYE

READY

  10 LET X=4
  20 LET Y=X↑2+3
  30 PRINT X,Y
  40 END
RUN
 4             19


READY
```

Observe: Obviously, care must be taken to insure that the variables have been given values **before** they are encountered in the program. That is, X must have a value before statement 20 can be executed.

```
 100 LET A=1.2
 110 LET B=6.3
 120 LET C=(A↑2+B↑2)↑(1/2)
 130 LET P=A+B+C
 140 LET X=(1/2)*A*B
 150 PRINT A,B,C,P,X
 160 END
RUN
 1.2           6.3           6.413267      13.91327      3.78

READY
```

Observe: Actually, the LET statements 120, 130, and 140 are not necessary, since the calculations involved could be made in the PRINT statement as in the example below:

```
 100 LET A=1.2
 105 LET B=6.3
 110 PRINT "SIDE A", "SIDE B", "SIDE C", "PERIMETER", "AREA"
 120 PRINT A,B,(A↑2+B↑2)↑(1/2),A+B+(A↑2+B↑2)↑(1/2),(1/2)*A*B
 130 END
RUN
SIDE A        SIDE B        SIDE C        PERIMETER     AREA
 1.2           6.3           6.413267      13.91327      3.78

READY
```

Observe: The column headings, by using commas, are centered over the five BASIC data columns.

Exercises:

1. What would be the appearance of the output of the following program?

```
100 LET M=4
200 LET N=3
300 PRINT M+N,M-N,M*N,M/N,M↑N
400 END
```

2. What would be the appearance of the output of the following program?

```
10 LET M=4
20 LET N=3
30 LET S=M+N
40 LET D=M-N
50 LET P=M*N
60 LET Q=M/N
70 LET E=M↑N
80 PRINT S,D,P,Q,E
90 END
```

3. Use the LET instruction in a program to determine the area of a rectangle, given a length of 5 inches and width of 2.68 inches.

4. Use the LET instruction in a program to determine the perimeter and area of a right triangle, given the perpendicular sides of 24 feet and 8.5 feet.

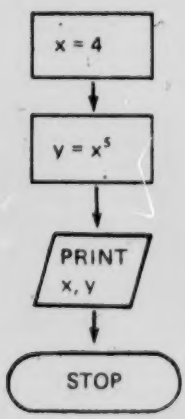
NOTE: Appendix I has a nice discussion about error messages--in case you have encountered them! Program modification is also discussed.

5. What is the output of the following program?

```
BYE

READY

 5 LET A=2
10 LET B=A+4
15 PRINT A,B
20 LET B=A+6
25 PRINT A,B
30 END
```

BRANCHING

The very simple programs illustrated thus far could be called "one-shot" programs because each statement is encountered just once. The following "flow chart" illustrates such a program:



This program would yield as output the values 4 and 1024.

```
  10 LET X=4
  20 LET Y=X↑5
  30 PRINT X,Y
  40 END
RUN
 4                1024

READY
```

Now, this program is not about to impress very many people. A simple calculator could obtain the value of 4^5 in much less time than it might take to "program" the problem on a computer.

So let's not postpone any longer the unveiling of some of the power of a computer. Much of the speed that we attribute to the computer is due to its ability to **process instructions as though they are numerical data** (because in a very real sense they are). A four-or five-step program is hardly a challenge for a computer which could process perhaps a hundred thousand in a second!

A greater challenge than the problem above might be that of obtaining the 5th power of all numbers 4 through 100. And a clever way of doing this might be to keep changing the value of "x".

Suppose, after printing x and y, we changed x to 5, then to 6, and so on.

```
10 LET X=4
20 LET Y=X↑5
30 PRINT X,Y
40 LET X=X+1
```

Whoa! What's this statement 40?

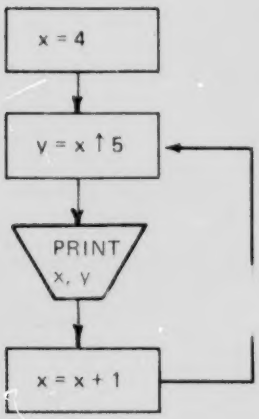
Well, the CPU obtains the value of x, adds 1 to it, and puts the new value back in the location called x. We are "incrementing" x. Now we could go back and find x ↑ 5 again (GO TO 20).

```
  10 LET X=4
  20 LET Y=X↑5
  30 PRINT X, Y
  40 LET X=X+1
  50 GO TO 20          <-------- Branch to Statement 20
  60 END
RUN
 4             1024
 5             3125
 6             7776
 7             16807
 8             32768
 9             59049
 10            100000
 11            161051
 12            248832
 13            371293
 14            537824
 15            759375
 16            1.048576E+6
```

21

How about that! Let's look at flow chart of this program.

```
 23            6·436343E+6
 24            7·962624E+6
 25            9·765625E+6
 26            1·188138E+7
 27            1·434891E+7
 28            1·721037E+7
 29            2·051115E+7
 30            2·430000E+7
```



We have created a "loop" in which statements 20, 30, 40, and 50 are encountered again and again (at fantastic speeds, too). This program will compute values of x^5 from 4 to ----------

```
 43            1·470084E+8
 44            1·649162E+8
 45            1·845281E+8
 46            2·059630E+8
 47            2·293450E+8
 48            2·548040E+8
```

Good heavens, when will it stop? Good question! On this particular machine, striking the "S" key stops a running program.

```
 84            4·182119E+9
 85            4·437053E+9
 86            4·704270E+9
 87
STOP

READY
```

Thing is – we could find an automatic way to stop when we are finished. But first, let's review the GO TO command.

GO TO Instruction

GO TO Format

_____ GO TO _____
(statement number) (statement number)

Characteristics:

1. This command causes a "branch" to the **statement number** referred to. It thus interrupts the sequential order in which program steps are executed by control.

2. This is an **unconditional** branch.

We will be using this instruction in a variety of ways, but let us return to our problem and the "endless" loop.

What we need to do is to test the value of x (or y) and decide when we have calculated all we had wished. Replace statement 50 with a conditional branch.

We could use the IF - THEN command instead of GO TO.

```
BYE

READY


  10 LET X=4
  20 LET Y=X↑5
  30 PRINT X,Y
  40 LET X=X+1
  50 IF X<26 THEN 20
  60 END
```

Observe: The IF - THEN command will cause a branch as long as x is not yet 26.

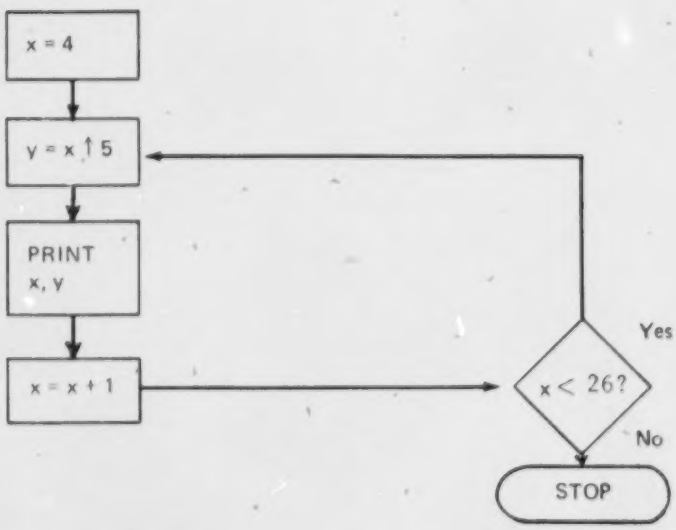
Now we have a **conditional** branch that will stop execution of our loop when a condition is met (x is not less than 26).

```
  10 LET X=4
  20 LET Y=X↑5
  30 PRINT X,Y
  40 LET X=X+1
  50 IF X<26 THEN 20
  60 END
RUN
```

(program continued on next page)

```
4             1024
5             3125
6             7776
7             16807
8             32768
9             59049
10            100000
11            161051
12            248832
13            371293
14            537824
15            759375
16            1.048576E+6
17            1.419857E+6
18            1.889568E+6
19            2.476099E+6
20            3.200000E+6
21            4.084101E+6
22            5.153632E+6
23            6.436343E+6
24            7.962624E+6
25            9.765625E+6

READY
```

Our flow chart now looks like this:



IF - THEN Format:

(statement no.)	IF	(expression)	(relation)	(expression)	THEN	(statement no.)

Characteristics:

1. The expressions referred to may be arithmetic expressions or variable names.
2. The **statement number** following the THEN is the statement to which control will branch if the condition is met.
3. If the condition is **not** met, the control **will step to the next instruction.**
4. The **relation** may be chosen from among the following:

Symbol	Meaning
=	equal to
>	greater than
<	less than
> =	greater than or equal to
< =	less than or equal to
< >	**not** equal to (some systems may use #)

Returning to the program, it should be noted that other choices of relations might have been used. The following programs provide the **same** output:

```
10 LET X=4
20 LET Y=X↑5
30 PRINT X,Y
40 LET X=X+1
50 IF X<=25 THEN 20
60 END
```

NOTE

```
10 LET X=4
20 LET Y=X↑5
30 PRINT X,Y
40 LET X=X+1
50 IF X>25 THEN 70
60 GO TO 20
70 END
```

```
READY

  10 LET X=4
  20 LET Y=X↑5
  30 PRINT X,Y
  40 LET X=X+1
  50 IF X=26 THEN 70
  60 GO TO 20
  70 END
```

NOTE

Since these programs provide the same output, it would be absurd to claim one program better than another. Purists would probably like the first (using $x < 26$). The machine could care less about an additional GO TO statement, which would take up very little additional memory.

A flow chart in advance would probably suggest the more "efficient" route.

Here are some examples of the use of the IF - THEN instructions.

```
BYE

READY

   5 PRINT "NUMBER", "SQUARE ROOT"
  10 LET A=1
  20 PRINT A,A↑.5
  30 LET A=A+1
  40 IF A<=10 THEN 20
  50 END
RUN
NUMBER          SQUARE ROOT
 1               1
 2               1.414214
 3               1.732051
 4               2
 5               2.236068
 6               2.44949
 7               2.645751
 8               2.828427
 9               3
 10              3.162278

READY
```

For Square Root

Observe: A variable or expression can be raised to fractional powers.

```
READY

   5 PRINT "X", "Y"
  10 LET X=-10 <-------------------- Initialize x
  15 LET Y=X↑3-6*X+2
  20 PRINT X,Y
  25 LET X=X+2
  30 IF X<=10 THEN 15 <------------- Branch until x > 10
  35 END
RUN
X             Y
-10           -938
-8            -462
-6            -178
-4            -38
-2             6
 0             2
 2            -2
 4             42
 6             182
 8             466
 10            942

READY
```

Observe: Here we are incrementing by 2. The function $y = x^3 - 6x + 2$ is evaluated from x = -10 to x = 10.

INPUT INSTRUCTION

The only method discussed thus far for entering data into a program is by including that data in the program by means of the LET and PRINT instruction. The INPUT instruction allows the user to enter data via a terminal.

INPUT Format

_____ INPUT _____
(statement number) (variable names)

Characteristics:

1. Any number of valid variable names may follow INPUT, each separated by commas.
2. The computer will type a "?" when encountering an INPUT instruction, to indicate readiness for data.

The following illustrates the use of the INPUT instruction in a simple program.

```
READY

  10 INPUT A,B
  20 LET X=5
  30 LET Z=(A+B)/X
  40 PRINT A,B,Z
  50 END
```

After the program has been stored in the memory of the computer, the instructions are executed in sequential order, unless a decision instruction is encountered. The first instruction which would be encountered above is statement 10, the INPUT instruction. The control unit tells the operator that the computer is waiting for a value for the variable **A** by typing a "?". The operator then enters a value via the terminal.

```
READY

  10 INPUT A,B
  20 LET X=5
  30 LET Z=(A+B)/X
  40 PRINT A,B,Z
  50 END
RUN
? 6
```

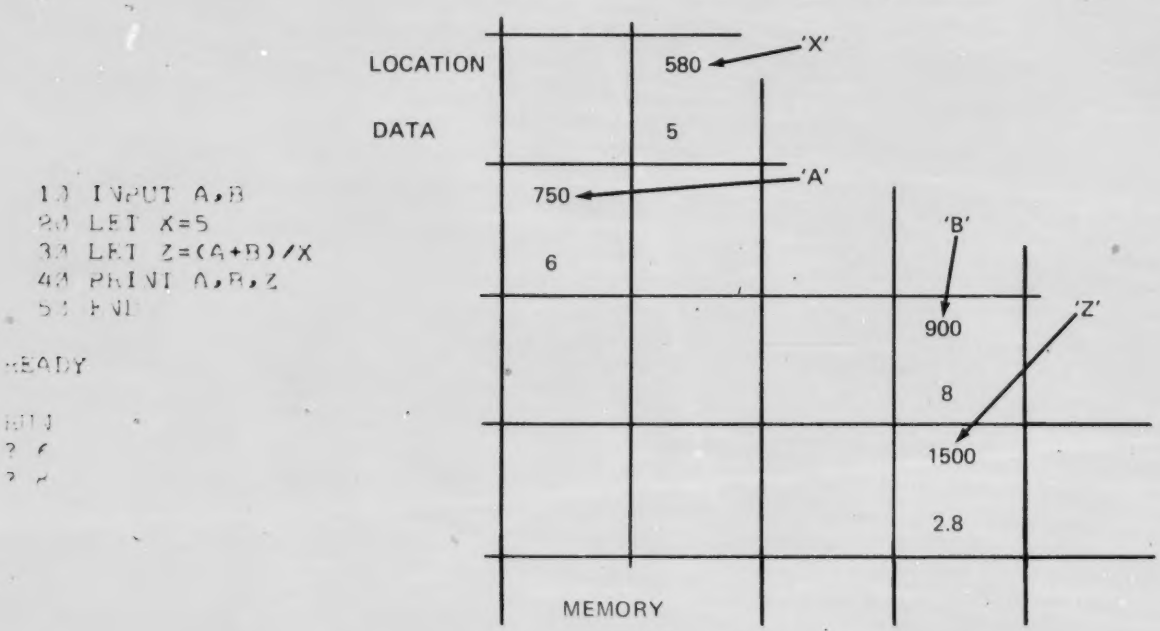
Operator enters a value for A

Computer has encountered the INPUT statement

The operator depresses the RETURN KEY or the COMMA KEY in order to tell the computer that the value has been entered. The computer scans the remaining space on the INPUT line, in order to determine if there are any more values to be entered. The computer may signal by typing another "?", indicating that another value needs to be entered. In this case the value of **B**. Once the value or values have been entered, depress the RETURN KEY and the program proceeds to function.

```
RUN
? 6
? 8
 6             8             2.8

READY
```

Suppose the value of **X** is stored in location 580. Any time the program refers to **X**, the software refers to the absolute or relative address of **X**. Suppose the area 750 and 900 are reserved for the variables **A** and **B**, respectively. Any data in locations 750 and 900 at the time that the INPUT instruction is executed are destroyed. The same is true when 5 is stored in location 580. This is termed **destructive read in.**



28

Let us step through the above program.

1. The INPUT instruction stores a value for **A** and **B** in 750 and 900, (when the operator enters these values).

2. The LET instruction stores a 5 in location 580 which can be referred to by the variable name **X**.

3. The LET instruction in statement number 30 adds the contents of locations 750 and 900 to an accumulator. The number in the accumulator then is divided by the contents of 580. Then the number in the accumulator is moved to location 1500, and address assigned to **Z**.

4. The PRINT instruction is then executed. The contents of locations 750, 900, and 1500 (i.e., A, B, **Z**) are printed.

Shown below is the completed program:

```
10 INPUT A,B
20 LET X=5
30 LET Z=(A+B)/X
40 PRINT A,B,Z
50 END
RUN
? 6
? 8
 6             8             2.8

READY
```

Observe: With this system, a comma after the first entered data did not cause another "?" to be printed. The other data was then entered.

The following programs illustrate the use of the INPUT statement:

```
  10 INPUT M,N
  20 PRINT M+N,M-N,M*N,M/N,M↑N
  30 END
RUN
? 6,8
 14           -2           48           .75          1.679616E+6

READY
```

```
  10 INPUT M,N
  20 PRINT M+N,M-N,M*N,M/N,M↑N
  30 GO TO 10
  40 END
RUN
? 6,8
 14           -2           48           .75          1.679616E+6
? 2.65,.8
 3.45         1.85         2.12         3.3125       2.180704
? -4,3
-1           -7           -12          -1.333333    -64
?
STOP

READY
```

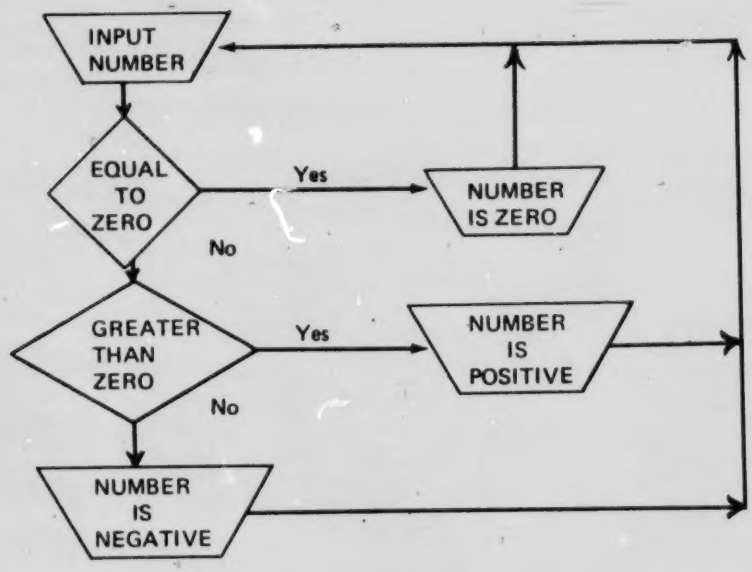

29

Observe: The GO TO branch causes control to return to the INPUT statement again and again. The program was eventually stopped by depressing "S" (or whatever procedure is provided by the system software.)

```
READY

10 INPUT N
20 IF N=0 THEN 80
30 IF N>0 THEN 60
40 PRINT "THE NUMBER IS NEGATIVE"
50 GO TO 10
60 PRINT "THE NUMBER IS POSITIVE"
70 GO TO 10
80 PRINT "THE NUMBER IS ZERO"
90 GO TO 10
100 END
RUN
? 5
THE NUMBER IS POSITIVE
? -71.893
THE NUMBER IS NEGATIVE
? 0
THE NUMBER IS ZERO
? 0.0000
THE NUMBER IS ZERO
? 0.00001
THE NUMBER IS POSITIVE
? -757575
THE NUMBER IS NEGATIVE
? 3.4E12
THE NUMBER IS POSITIVE
?
STOP

READY
```

Observe: Only statements 10, 20, 80, and 90 are executed when zero is input. Positive numbers cause statements 10, 20, 30, 60, and 70 to be executed. Statements 10, 20, 30, 40, and 50 are executed when negative numbers are input. Notice the last data input is in exponential form.

The following is a flow chart of the problem printed above.



There is another command in the BASIC vocabulary which, although unnecessary, may be useful in adding clarity to a program. The command STOP will cause an end to the execution of a program. There can be just one END statement, but we use as many STOP commands as we wish. **It is equivalent to a GO TO branch to the END statement.** The following are equivalent:

```
READY

.........
.........
.........
65 GO TO 110
.........
.........
.........
110 END
```

```
....
....
65 STOP
....
...
...
110 END
```

Exercises:

1. What is the output of the following program?

```
10 LET S=0
20 LET A=1
30 LET S=S+A
40 LET A=A+1
50 IF A<=10 THEN 30
60 END
```


31

2. What is the output of the following program?

```
10 LET S=0
20 LET N=1
30 LET S=S+N
40 LET N=N+1
50 IF N<=10 THEN 30
60 PRINT S
70 END
```

3. What is the output of the following program?

```
10 LET N=0
20 LET A=1
30 PRINT A
40 LET R=A
50 LET A=A+N
60 LET N=R
70 IF A<10 THEN 30
80 END
```

4. Write a program to determine the values of the function $y = x^2 - 4x + 5$ for $x = -10, -9, -8, \ldots 9, 10$.

5. Write a program to determine the values of the function $y = 2x^3 - 4x + 5$ for input values of x. Provide a branching instruction for repeated entry of data.

6. Write a program to determine the perimeter and area of a right triangle, with the length of the two perpendicular sides given. Use the INPUT statement to enter the lengths of the sides. Print the length of the sides, the perimeter, and area.

7. Write a program that will produce a table of squares, square roots, cubes, and cube roots of the integers 1 - 25. Print the number in column 1, squares in column 2, etc. Print column headings.

8. Write a program to produce a table showing the amount (value) of an investment of $10,000 compounded annually at 6%. Print columns indicating the year, interest earned, and value of the investment.

UNIT III

READ AND DATA STATEMENTS

As an alternative to the INPUT statement, which assigns data to variables via the terminal **during** program execution, the READ-DATA statements can be used. The READ and DATA statements provide the assignment of data to variables by means of a data list **included in the program.** Therefore, there is no interruption of the program during execution.

READ, DATA Format:

_____ READ _____
(statement number) (variables names separated by commas)

_____ DATA _____
(statement number) (list of numerical data)

Note: These two instructions are given together since both must be present in a program if either is present.

Characteristics:

1. READ is an executable statement, that is, the instruction causes the computer to perform an act. When the statement is encountered, the variable names will be assigned values from the DATA list.

2. DATA is a non-executable statement and **can be placed anywhere in the program** preceding the END statement. The items in the list following DATA are assigned to the variables as they are encountered in the READ statement.

Following are two programs that function in the same manner except for the method of entering values:

```
 100 INPUT A,B,C
 110 LET X=A+B+C
 120 LET Y=X↑2
 130 PRINT A,B,C,X,Y
 140 GO TO 100
 150 END
RUN
? 5,4,2
 5             4             2             11            121
? 8,9,-1
 8             9            -1             16            256
? 8.3,-6.1,10
 8.3          -6.1           10            12.2          148.84
?
STOP

READY
```

```
 100 READ A,B,C
 110 LET X=A+B+C
 120 LET Y=X↑2
 130 PRINT A,B,C,X,Y
 140 GO TO 100
 150 DATA 5,4,2,8,9,-1,8.3,-6.1,10
 160 END
RUN
 5             4             2             11            121
 8             9            -1             16            256
 8.3          -6.1           10            12.2          148.84

ERROR  47 IN  100

READY
```

Observe: The second program, using the READ and DATA statements will assign –

A = 5 , B = 4 , C = 2 ,

then A = 8 , B = 9 , C = -1 ,

then , A = 8.3, B = -6.1, C = 10,

and automatically terminate the program upon exhausting the data. **The error message announces the end of data.**

Perhaps the choice of the INPUT or the READ - DATA statements ultimately depends on whether or not the programmer knows which data he/she wishes to process at the time of writing the program. Of course, a DATA statement can be modified whenever the program has been located, just as any statement may be modified.

Below are further examples of the use of the READ and DATA statements.

```
 100 READ A,B
 110 PRINT A,B,A+B,A-B,A*B
 120 DATA 4,2,6.1,3,-3,4,8,6
 130 DATA 11,3,5,2,-5,6.34,5.34E6,3
 140 GO TO 100
 150 END
RUN
 4             2             6             2             8
 6.1           3             9.1           3.1           18.3
-3             4             1            -7            -12
 8             6             14            2            48
 11            3             14            8            33
 5             2             7             3            10
-5             6.34          1.34         -11.34        -31.7
 5.340000E+6   3             5.340003E+6   5.339997E+6   1.602000E+7

ERROR  47 IN  100

READY
```

Observe: Two DATA statements are used and may be placed anywhere in the program. The READ statement here commands the control unit to take **two** values of data from the DATA list **each time the READ statement is encountered.**

Observe: Again, the error message indicates that data is exhausted.

It may be that we do not want the program stopped after exhausting the data. The following program uses an IF — THEN statement to detect the end of data and cause a branch at that time.

```
 10 LET S=0
 20 READ A
 30 DATA 13,4,2,17,4,6,7.5,23,56.7,-99 <------- FLAG
 60 IF A=-99 THEN 90 <------------------------- Test for Flag
 70 LET S=A+S
 80 GO TO 20
 90 PRINT S
130 END
RUN
 133.2

READY
```

Observe: Statements 10 through 80 are repeated (a loop) 9 times, once for each of the 9 data items encountered by the READ command. After the READ A assigns -99 to A, statement 60 causes a branch to statement 90 and the remainder of the program.

Question: What does S represent? Does this include -99?

Observe: Statements 10 and 70 allow us to sum the data list!

SUMMING AND COUNTING

We have seen how a data item may be used to signal the end of a data list and cause control to branch to the remainder of the program. This technique may be used to sum groups of data.

```
 10 LET S=0
 20 READ A
 30 DATA 13,4,2,17,4,6,7.5,23,56.7,-99
 40 DATA 5,3,10,7,8,16,23,14.5,-99,1,2,3
 50 DATA 4,5,6,7,8,9,10,-99 <------- FLAG
 60 IF A=-99 THEN 90
 70 LET S=A+S
 80 GO TO 20
 90 PRINT "SUM =" S
100 GO TO 10
130 END
RUN
SUM = 133.2
SUM = 86.5
SUM = 55

ERROR  47 IN   20
```

Observe: The data item -99 causes the PRINT of the current value of the sum S. Then statement 100 returns control to statement 10 which resets the sum S to 0 (in preparation for summing another list of data).

We may wish to expand the program to obtain the sum of **all data**. The following additions to the program will accomplish this purpose.

```
READY

5 LET T=0
50 DATA 4,5,6,7,8,9,10,-999
55 IF A -999 THEN 110
95 LET T=T+S
110 PRINT "SUM=" S
120 PRINT "TOTAL"=" T
115 LET T=T+S
```

A LIST command will provide the modified program.

```
5 LET T=0
10 LET S=0
20 READ A
30 DATA 13,4,2,17,4,6,7·5,23,56,0 + + ·7,-99
40 DATA 5,3,10,7,8,16,23,14·5,-99,1,2,3,
50 DATA 4,5,6,7,8,9,10,-999 <------------ End of all data
55 IF A = -999THEN 110
60 IF A = - 99 THEN 90
70 LET S=A+S
80 GO TO 20
90 PRINT "SUM =" S
95 LET T=T+S
100 GO TO 10
110PRINT "SUM=" S
115 LET T=T+S
120 PRINT "TOTAL=" T
130 END

READY

RUN
SUM = 133·2
SUM = 86·5
SUM = 55
TOTAL= 274·7

READY
```

Observe: Statement 55 detects -999 and causes execution of 110, 115, and 120 for a PRINT of the last S and the total summation T. In this program, T is obtained by summing the 3 values of S. We **could** have obtained a T by summing **all** A.

We may further embellish (complicate) this program by obtaining a **count** of the number of data items. These additions are necessary:

36

```
15 LET C = 0
75 LET C=C+1
90 PRINT "SUM =" S, "COUNT =" C
105 LET C=C+1
110 PRINT "SUM =" S, "COUNT =" C

SCR
```

Observe: Statement 75 increments C by 1, so that we may **count** the data items.

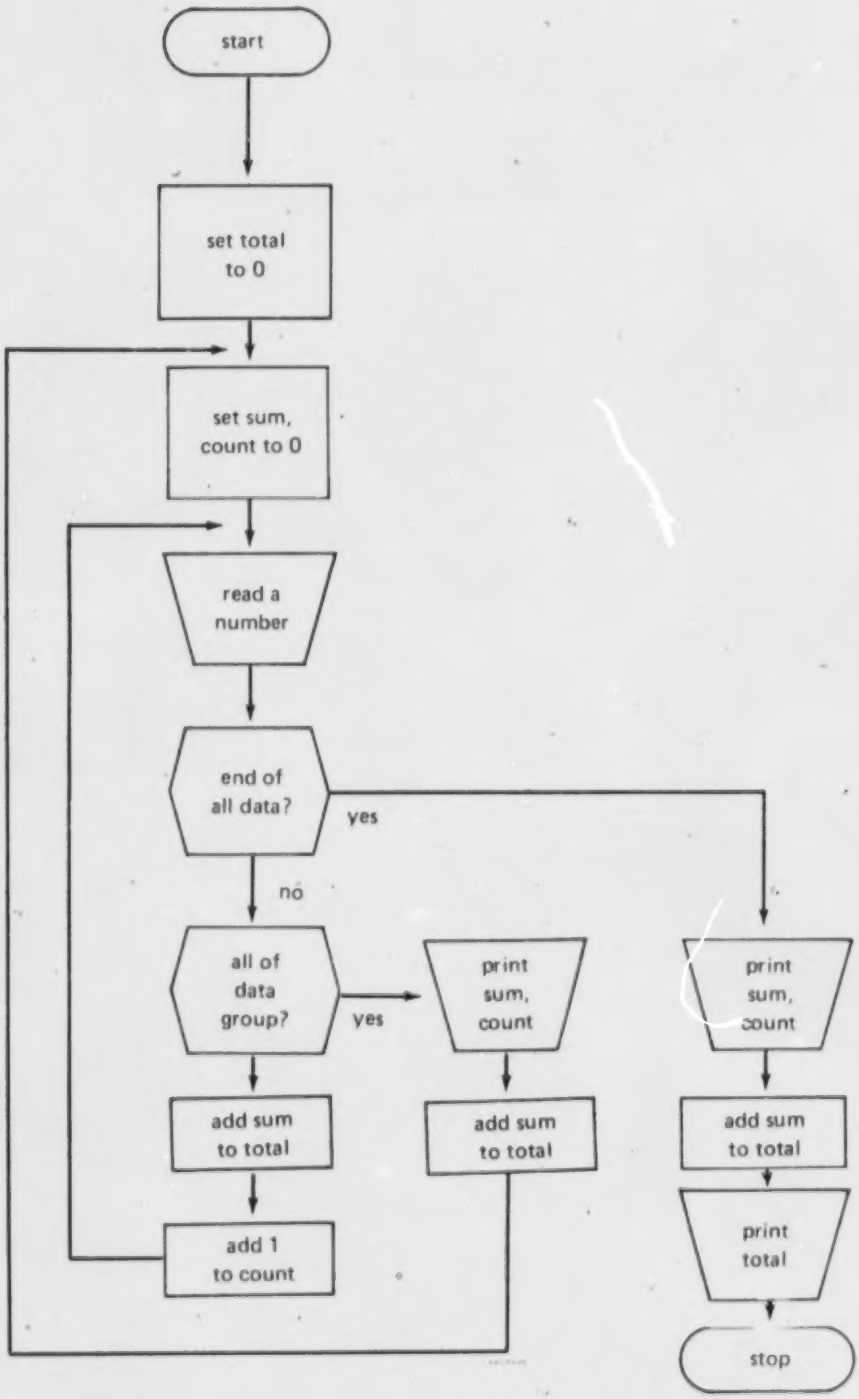
Once again, a LIST of the modified program:

```
  5 LET T=0
 10 LET S=0
 15 LET C=0
 20 READ A
 30 DATA 13,4,2,17,4,6,7.5,23,56.7,-99
 40 DATA 5,3,10,7,8,16,23,14.5,-99,1,2,3
 50 DATA 4,5,6,7,8,9,10,-999
 55 IF A=-999 THEN 110
 60 IF A=-99 THEN 90
 70 LET S=A+S
 75 LET C=C+1 <------------------------- For Count
 80 GO TO 20
 90 PRINT "SUM =" S, "COUNT =" C
 95 LET T=T+S
100 GO TO 10
105 LET C=C+1 <------------------------- For Count
110 PRINT "SUM =" S, "COUNT =" C
115 LET T=T+S
120 PRINT "TOTAL=" T
130 END
RUN
SUM = 133.2    COUNT = 9
SUM = 86.5     COUNT = 8
SUM = 55       COUNT = 10
TOTAL= 274.7

READY
```

Observe: Further additions to the program could obtain a **total count** of the data items.

Recognize that there is a variety of ways of obtaining sums and counting. There is probably no best way, but the above programs should have illustrated some general principles.

On the following page is a flow chart of the above problem.


start
set total to 0
set sum, count to 0
read a number
end of all data?
yes
no
all of data group?
yes
print sum, count
print sum, count
add sum to total
add sum to total
add sum to total
add 1 to count
print total
stop

38

Exercises:

1. Write a program that will find the sum and count of a list of numbers. Use the READ-DATA statements.

2. Write a program that will find the sum and average of a list of numbers. Use the READ-DATA statements.

3. Write a program that will find the sum of all integers from 1 to 100.

4. Write a program to find the sum of the first 10 terms of the series 1+1/2+1/3+1/4+.....+1/n.

5. Modify the program of exercise 4 so as to find the sum of the first 20, 30, and 40 terms of the series (in addition to the first 10 terms).

FUNCTIONS

The various computer manufacturers have supplied standard function sub-routines, which are supposed to make the programmer's life easier. The BASIC functions often available are:

Function	Computer determines -
SQR (X)	The square root of X
LOG (X)	the natural log of X
EXP (X)	e^X (actually 2.718282^X)
ABS (X)	the absolute value of X
INT (X)	the greatest integer **less than** X
SIN (X)	the sine of X, "where X is in radians"
COS (X)	the cosine of X, "where X is in radians"
TAN (X)	the tangent of X, "where X is in radians"
ATN (X)	the arc tangent of X, "where X is in radians"
RND (X)	a non-statistical pseudo-random number between 0 and 1.

Most, if not all, of these functions will be available with the BASIC you will use.

These functions may be used in LET, PRINT, and IF-THEN statements (and other statements to be taken up later) in the same way that arithmetic expressions are used.

The argument "X" may be a single variable name or an arithmetic expression. Study the following programs:

```
READY

100     INPUT A
200     PRINT SQR(A)
300     GO TO 100
400     END
RUN
? 1
 1
? 2
 1.414214
? 3
 1.732051
? 4
 2
? 5
 2.236068
?
STOP

READY
```

```
  10 INPUT A
  20 PRINT SQR(A+3)
  30 GO TO 10
  40 END
RUN
? 1
 2
? 2
 2.236068
? 3
 2.44949
? 4
 2.645751
? 5
 2.828427
? 6
 3
?
```

```
  10 REM PART OF SUBROUTINE FOR DIVISIBILITY CHECK
  20 INPUT A,D
  30 LET X=A/D
  40 LET Y=INT(A/D)
  50 PRINT X,Y
  60 REM IF X=Y THEN D DIVIDES A
  70 GO TO 20
  80 END
RUN
? 6.4,2
 3.2           3
? 9,3
 3             3
? 693,3
 231           231
? 161,7
 23            23
? 46934,8
 5866.75       5866
?
```

Observe: Notice that if X = Y, then D divides A. This comparison of a quotient and the integer part of the quotient will serve as a check of divisibility!!!

```
 10 PRINT "ANGLE (DEG)", "SIN N", "COS N", "TAN N"
 20 LET N=0
 30 LET A=N*3.14159/180
 40 PRINT N, SIN(A),COS(A),TAN(A)
 50 LET N=N+30
 60 IF N<370 THEN 30
 70 END
RUN
ANGLE (DEG)    SIN N           COS N           TAN N
 0              0               1               0
 30             .4999996        .8660256        .5773497
 60             .866025         .5000008        1.732047
 90             1               1.310775E-6     762907.7
 120            .8660263       -.4999985       -1.732058
 150            .5000019       -.8660243       -.5773532
 180            2.668363E-6    -1              -2.668363E-6
 210           -.4999973       -.8660269        .5773462
 240           -.8660236       -.5000031        1.732037
 270           -1              -3.979137E-6     251310.8
 300           -.8660276        .4999962       -1.732069
 330           -.5000042        .8660229       -.5773568
 360           -5.289912E-6     1              -5.289912E-6

READY
```

Observe: Notice that the argument for trigonometric functions must be in radians. Statement 30 converts degree measure to radian measure.

Observe: Recognize that the computer must deal with rational approximations of repeating decimals and irrational numbers. The small numbers (in exponential form) in the output are therefore approximations for zero. Notice the other differences between these tables and the theoretical values of the trigonometric functions.

On the next page is a program for printing a list of random numbers from 1 through 30:

```
 10 REM J. STOUDT 13/40/84
 20 REM RANDOM NUMBERS 1 TO 30
 30 PRINT INT(1+30*RND(X))
 40 GO TO 30
 50 END
RUN
 8
 9
 22
 10
 10
 2
 15
 15
 16
 2
 8
 10
 6
 28
 8
 30
 8
 1
 27
 28
 25
 9
 12
 7
8
STOP

READY
```

Observe: An argument for RND is not given since the computer generates this number. The RND function in this system generates randoms from 0 to 1. Multiplying by 30 gives randoms from 0 to 30. Adding 1 gives us random numbers from 1 to 31. The INT function gives us integer values from 1 through 30.

Exercises:

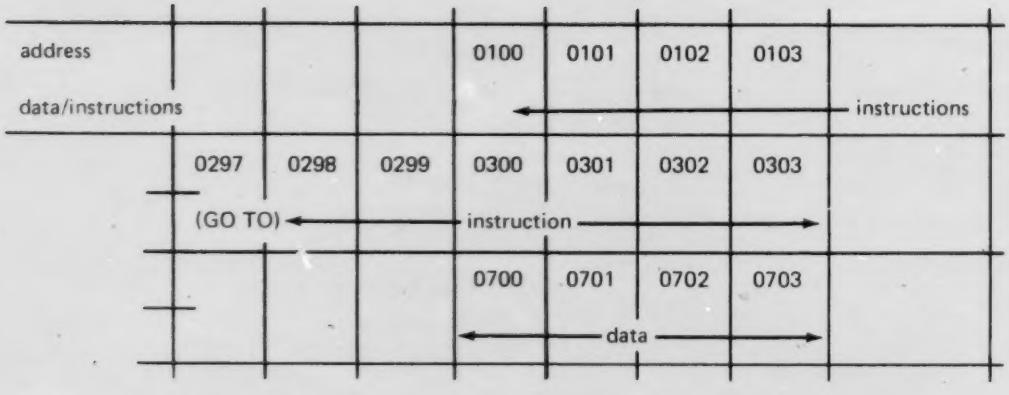
1. Write a program to check the divisibility of an INPUT number by 3.
2. Write a program which, if given a positive decimal number, will round that number off to the nearest one-hundredth. Hint: You can truncate a positive number by using the INT function.
3. Write a program which will generate random numbers from 1 to 52. What might this program be used for?
4. Write a program of your choosing which employs other of the built-in functions.
5. Write a program to find the quotient and remainder of an INPUT number divided by 7.

THE KEY TO SPEED

The digital computer gets much of its power and speed from its ability to store and process program steps just as it stores and processes data. The program, stored sequentially in the memory unit, is in machine language and as such has the same appearance as numerical data and can be processed with the same kind of speed.

The program steps have addresses as do other data and it is the address which ultimately distinguishes instruction from data. The control unit does arithmetic on the addresses of instructions just like the arithmetic/logic unit does arithmetic on data.

Let us say that in a hypothetical machine, a given program is stored sequentially in addresses 0100 - 0303. The control unit, given the RUN command, will fetch these instructions one at a time, starting at address 0100 and proceeding to 0101, 0102, etc.



It is an easy matter at some point in the program, let us say the instruction at address 0297, to instruct the control unit to go back to 0150, or **forward** to 0301. This is the object of the GO TO instruction discussed in Unit II.

Suppose a READ instruction in the program was located in 0170-0174 (taking up more than one machine word). These machine instructions might command the control to go to the data string starting at 0700 and begin fetching this data. In some machines this data string may be **within** the program instruction string. In either case, data would be obtained as required by the READ instruction.

Remember that the control unit of the computer, through the use of a register, keeps track of where it is in the program, fetching instructions from memory and acting upon them.

THE FOR AND NEXT STATEMENTS

The decision-making IF-THEN statement has been used in perhaps two different ways: 1) to branch into one process or another depending upon the condition tested, and 2) to get out of a repetitive (loop) process upon meeting a certain condition. There is a pair of BASIC statements which serves the latter purpose well and is perhaps more useful in certain instances. The following program structures are equivalent.

```
10 LET N=3                    10 FOR N=3 TO 19 STEP 2
15.....                       15 .....
20 ...                        .
.                             .
.                             .
.                             .
.                             .
.                             .
60 LET N=N+2                  60 NEXT N
65 IF N<21 THEN 15            65 ...
70 .....
```

Observe: In either program, three things are done to control the loop:

1. Initialize N----Statement 10 on the left and "FOR N = 3" on the right.

2. Increment N----Statement 60 on the left and the "STEP 2" on the right.

3. Stop the loop----the IF-THEN on the left and the "TO 19" on the right.

Observe: The "NEXT N" statement is equivalent to a GO TO statement. "NEXT N" branches back to the FOR statement where N assumes a new value. Once N has assumed its final value (19), the computer falls through the "NEXT N" statement to statement 65 which follows.

FOR-NEXT Format

_____	FOR	_____	=	_____	TO	_____	STEP	_____
(st. no.)		(var. name)		(constant, var. name, expression)		(constant, var. name, expression)		(constant, var. name, expression)

_____	NEXT	_____
(st. no.)		(variable name)

Characteristics

1. This statement is used to set up program loops and iterations. The "FOR---" statement defines the **beginning** of the loop; "NEXT---" defines the **end** of the loop.

2. Variables and expressions used in the "___TO___STEP___" phrase must of course have numerical values when this statement is encountered.

The following programs illustrate the use of the FOR, NEXT statements.

```
   5 LET B=2
   7 REM J.DOE--CALCULATES POWERS OF 2 FROM 1 TO 10
  10 FOR A=1 TO 10 STEP 1
  30 LET X=B↑A
  40 PRINT "2↑"; A,X
  50 NEXT A
  60 END
RUN
2↑ 1            2
2↑ 2            4
2↑ 3            8
2↑ 4            16
2↑ 5            32
2↑ 6            64
2↑ 7            128
2↑ 8            256
2↑ 9            512
2↑ 10           1024

READY
```

Observe: The FOR statement in the above program (statement 10) sets A equal to 1 and increments A's value by what follows the step section of the FOR statement. In this case it is incremented by 1 until A's value is greater than 10. "STEP 1" could be omitted in this program and the computer would assume an increment of 1.

```
  15 REM J. DOE 8/5/84
  20 REM POWERS OF NUMBERS 1 THRU 15 USING FOR-NEXT LOOP
  25 FOR N=1 TO 15
  30 PRINT N,N↑2,N↑3,N↑4,N↑5
  35 NEXT N
  40 END
RUN
 1              1              1              1              1
 2              4              8              16             32
 3              9              27             81             243
 4              16             64             256            1024
 5              25             125            625            3125
 6              36             216            1296           7776
 7              49             343            2401           16807
 8              64             512            4096           32768
 9              81             729            6561           59049
 10             100            1000           10000          100000
 11             121            1331           14641          161051
 12             144            1728           20736          248832
 13             169            2197           28561          371293
 14             196            2744           38416          537824
 15             225            3375           50625          759375

READY
```

Observe: We obtained an increment of 1 for N by default, since "STEP 1" was omitted.

```
  15 REM J. DOE 8/5/70
  20 REM POWERS OF ODD NUMBERS 1 THRU 15 PRINTED IN REVERSE ORDER
  25 FOR N=15 TO 1 STEP -2
  30 PRINT N,N↑2,N↑3,N↑4,N↑5
  35 NEXT N
  40 END
RUN
 15            225           3375          50625         759375
 13            169           2197          28561         371293
 11            121           1331          14641         161051
 9             81            729           6561          59049
 7             49            343           2401          16807
 5             25            125           625           3125
 3             9             27            81            243
 1             1             1             1             1

READY
```

Observe: Here we have stepped "down" from 15 to 1 by an increment of 2.

As our programs become more complex, the need for the "nested" FOR-NEXT may arise. The following illustrates one FOR-NEXT loop nested within another.

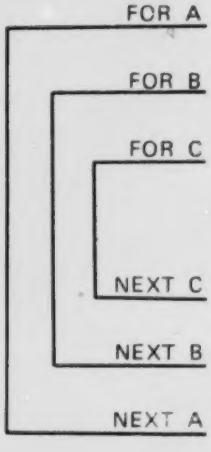
```
  20 FOR D=1 TO 3
  30 FOR R=1 TO 5
  40 PRINT "D=" D, "R=" R
  50 NEXT R
  60 NEXT D
  70 END
RUN
D= 1          R= 1
D= 1          R= 2
D= 1          R= 3
D= 1          R= 4
D= 1          R= 5
D= 2          R= 1
D= 2          R= 2
D= 2          R= 3
D= 2          R= 4
D= 2          R= 5
D= 3          R= 1
D= 3          R= 2
D= 3          R= 3
D= 3          R= 4
D= 3          R= 5

READY
```

Observe: The variable R took values of 1 to 5 for each of the 3 values of D.

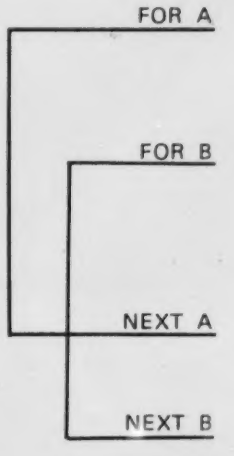
Nesting is allowed as long as the field of one loop does not cross the field of another.

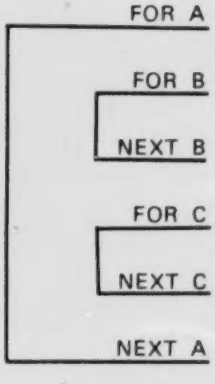
The following illustrates acceptable and unacceptable nesting techniques.

Acceptable



Unacceptable





What is the value of T derived of the following program?

```
 5 LET T=0.0
10 FOR K=1 TO 4
20 FOR L=1 TO 10
30 FOR M=1 TO 100
40 LET T=T+1
50 NEXT M
60 NEXT L
70 NEXT K
80 PRINT T
90 END
```

48

If you think T has a value of 4000 when the program has been run, you are correct.

Observe: The variable T in this program is a tally of the number of times the LET statement has been executed in the **inner** loop.

Observe: Statement 5 assures us that the tally T has an initial value of zero when we enter the program. **This is a worthwhile and often a necessary practice.** The address that has been given the label T could conceivably have a value other than zero when the program is entered.

Exercises:

1. What is the output of the following programs?

```
10 FOR X=-5 TO 5 STEP 2
20 PRINT X
30 NEXT X
40 END
```

```
10 FOR X=-5 TO 5 STEP 3
20 PRINT X
30 NEXT X
40 END
```

```
10 FOR X=-5 TO 5 STEP 4
20 NEXT X
30 PRINT X
40 END
```

2. Write a program, using the FOR and NEXT statements, to evaluate $4x^2 - 2x + 3$ for integral values of x from -5 to 5.

3. Rewrite the following program, using FOR and NEXT statements to control the loop.

```
10 LET X=3
20 PRINT 2*X↑2-5*X-8
30 LET X=X+2
40 IF X<16 THEN 20
50 END
```

4. Rewrite the following program **without** using the FOR and NEXT statements.

```
10 FOR X=4 TO -12 STEP -4
20 PRINT X
30 NEXT X
40 END
```

5. Write a program of your own choosing, employing the FOR and NEXT statements.

UNIT IV

DEFINING A FUNCTION

There may be instances when the programmer will want to compute a rather long expression (function) in different steps of the program, and would like to save a little space (or typing).

Suppose we were evaluating functions like

$$y = 2x^3 - 3x^2 + 4x - 5 \quad \text{and} \quad f(a)\ 3a^5 - a^2 + 9$$

for a variety of values of x and a in a program. We may simply define these functions by statements like the

```
99 DEF FNA(X) 2*X↑3-3*X↑2+4X-5

76 DEF FNB(A)= 3*A↑5-A↑2+9
```

We may now make use of these functions just as we have used the built-in functions. Below are three statements which illustrate how we may call for values of the functions defined above.

```
60 LET M = FNA(.1)

64 LET M=FNA(3.1)

14 PRINT X,Y,FNB(2), FNA(-3)

1019 IF C=FNB(4) THEN 50
```

DEF FN Format

(statement no.) **DEF FN** (function name) ( ) = (variable) (algebraic expression)

Characteristics:

1. The function may be named by **one** letter A to Z.
2. The variable must be the same used in the algebraic expression. Any letter A to Z may be used. (It is a **dummy** variable and need not be referred to anywhere else in the program.)
3. To obtain the use of the value of this function elsewhere in the program, we simply use FN ( ) with the proper name and desired arguments just as we would for the built-in functions.
4. Other functions and real variables may appear in the algebraic expression.
5. Some BASIC software packages allow a function to be defined in two or more variables like

DEF FNM(X, Y) = X + Y • 3

The following programs illustrate the **use** of this statement. Obviously, these programs are too simple to demonstrate the **need** of this statement.

```
10 DEF FNC(B)=10*B↑4-3.5*B↑2-9.5
20 LET Y=-5
30 PRINT Y, FNC(Y)
40 LET Y=Y+1
50 IF Y=<=5 THEN 30
60 END
RUN
-5              6153
-4              2494.5
-3              769
-2              136.5
-1             -3
 0             -9.5
 1             -3
 2              136.5
 3              769
 4              2494.5
 5              6153

READY
```

Observe: Notice that **B** is a dummy variable.

Observe: If you are familiar with polynomial zeroes, notice that the **value** of the function changes signs twice, i.e., we passed two zeroes.

```
12 REM J. STOUDT --- COMPOSITION OF FUNCTIONS
16 PRINT "X", "F(X)=2X+7","G(X)=1/2(X-7)","F(G(X))","G(F(X))"
20 DEF FNF(A)=2*A+7
24 DEF FNG(A)=0.5*(A-7)
28 LET N=-5
32 LET S=FNF(N)
36 LET T=FNG(N)
40 PRINT N,S,T,FNF(T),FNG(S)
44 LET N=N+1
48 IF N<=5 THEN 32
52 END
RUN
X              F(X)=2X+7      G(X)=1/2(X-7) F(G(X))        G(F(X))
-5             -3             -6             -5            -5
-4             -1             -5.5           -4            -4
-3              1             -5             -3            -3
-2              3             -4.5           -2            -2
-1              5             -4             -1            -1
 0              7             -3.5            0             0
 1              9             -3              1             1
 2              11            -2.5            2             2
 3              13            -2              3             3
 4              15            -1.5            4             4
 5              17            -1              5             5

READY
```

52

Observe: Statement 28 serves to initialize **N**.

Exercise:

1. Write a program of your own choosing which employs the DEF FN statement.

SUBSCRIPTED VARIABLES

Virtually all BASIC software packages allow the use of variable names such as A2 or X3 as well as the simpler names like A, B, M, or Y. There are circumstances, however, where we would like to retain data in an array or matrix. Subscripted variables such as A(6) or X(9) will prove advantageous for this purpose. BASIC allows us to use a **variable name for the subscript** and thus represent elements of an array or matrix with designations such as A(I) or C(X, Y).

The following statements will read in and assign labels to 10 pieces of data:

```
10 DIM X(9)
20 FOR I=0 TO 9
30 READ X(I)
40 NEXT I
50 DATA 5,3,10,15,-25,35,-3,25,81,17
```

Observe: The DIM statement reserves memory for the 10-member array labeled X(0), X(1), X(2), ..., X(9).

Observe: The FOR and NEXT statements supply **subscript values** for I in the READ X(I) statement.

After these steps have been executed, the memory will look like this:

(address labels)	X(0)	X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)	X(9)
data	5	3	10	15	-25	35	-3	25	81	17

Similarly, the following statements would provide **output** of the elements of this array.

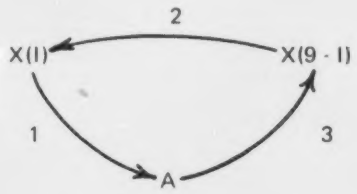
```
 110 FOR I=0 TO 9
 120 PRINT X(I)
 130 NEXT I
 140 END
RUN
 5
 3
 10
 15
-25
 35
-3
 25
 81
 17

READY
```

The advantage of the subscripted variable is that a single designation such as X(I) can be used to reference an entire array.

Suppose we wished to reverse the order of this data list in memory. These statements would accomplish that purpose:

```
 60 FOR I=0 TO 4
 70 LET A=X(I)
 80 LET X(I)=X(9-I)
 90 LET X(9-I)=A
100 NEXT I
```

Observe: The variable A is the label for an address used to "hold" X(I) while X(I) takes on the value of X(9 - I). Schematically:



The following statements read INPUT values into a 3 x 5 matrix:

```
10 DIM M(2,4)
20 FOR R=0 TO 2
30 FOR C=0 TO 4
40 INPUT M(R,C)
50 NEXT C
60 NEXT R
```

Observe: The **first** number entered by INPUT would be stored in the address labeled M(0, 0); the **second** would be labeled M(0, 1); etc.,
then M(1, 0), M(1, 1), M(1, 2), M(1, 3), M(1, 4),
then M(2, 0), M(2, 1), M(2, 2), M(2, 3), M(2, 4).

Dimension Statement

The DIM statement defines the **maximum number** of elements in an array. The elements of the array are designated by subscripted variables.

Format:

_____ DIM ____ (____)
(st. no.) (var.) (subsc.)

or

_____ DIM ____ (_____ , _____)
(st. no.) (var.) (S1) (S2)

54

Characteristics:

1. This statement reserves space in memory for an array of data. It defines the maximum number of elements in the array.

2. A dimension statement of DIM X(4) reserves 5 addresses labeled X(0), X(1),. . ., and X(4) for data.

Here are some examples of the DIM statement:

10 DIM A(3, 4) (reserves space for a 4 x 5 matrix of 20 elements)

40 DIM C(10) (reserves space for 11 elements designated C(0), C(1), . . ., C(10))

5 DIM X(8, 8) (reserves space for a matrix of elements designated X(0, 0) to X(8, 8))

The following programs make use of subscripted variables and the DIM statements associated with them.

```
10 REM READ AN ARRAY INTO MEMORY
20 REM AND PRINT IT OUT IN REVERSE ORDER
30 DIM L(5)
40 FOR I=1 TO 5
50 INPUT L(I)                    ---- Input Array
60 NEXT I
70 FOR B=5 TO 1 STEP -1
80 PRINT L(B)                    ---- Output Array
90 NEXT B
100 END
```

Observe: Recognize that in this program the subscripted variable L(I) is not actually necessary, but it does allow the storage of 5 pieces of data before the processing (printing out) of that data. The reversal of order is accomplished by printing out the array in reverse order. In a previous example, the data were reversed **in memory**.

```
  5 REM AVERAGE OF 5 GRADES FOR EACH OF 30 STUDENTS
 10 DIM G(30,5),A(30)
 15 PRINT "STUDENT", "SCORES",, "AVERAGE"
 20 REM S STANDS FOR THE STUDENT NUMBER
 30 FOR S=1 TO 30
 40 REM T STANDS FOR THE TEST SCORE
 50 FOR T=1 TO 5
 60 READ G(S,T)
 70 LET A(S)=A(S)+G(S,T)
 80 REM AT THIS POINT A(S) ACCUMULATES SCORES
 90 NEXT T
100 LET A(S)=A(S)/5
110 REM A(S) IS NOW THE AVERAGE
120 NEXT S
130 FOR S=1 TO 30
140 PRINT S,G(S,1);G(S,2);G(S,3);G(S,4);G(S,5), A(S)
145 NEXT S
150 DATA 80,85,84,75,86,55,64,75,80,84,90,85,88,95,86
160 DATA 76,78,82,80,65,74,80,70,85,88,96,94,100,95,90
170 DATA 90,85,86,95,84,65,70,55,60,75,80,82,88,75,86
180 DATA 50,55,70,65,55,85,90,92,88,94,90,92,88,95,95
190 DATA 80,85,82,81,90,82,80,86,80,90,95,98,95,100,100
200 DATA 86,95,88,85,90,84,80,75,64,55,86,75,84,85,80
210 DATA 80,85,90,85,82,70,75,65,78,70,100,95,100,96,100
220 DATA 85,85,80,90,92,90,95,92,90,96,70,60,75,68,50
230 DATA 90,95,100,94,96,88,85,70,80,74,65,80,82,78,76
240 DATA 85,88,84,86,90,98,100,100,95,100,75,80,85,85,90
250 END
```

```
RUN
STUDENT        SCORES                          AVERAGE
 1              80  85  84  75  86              82
 2              55  64  75  80  84              71.6
 3              90  85  88  95  86              88.8
 4              76  78  82  80  65              76.2
 5              74  80  70  85  88              79.4
 6              96  94  100  95  90             95
 7              90  85  86  95  84              88
 8              65  70  55  60  75              65
 9              80  82  88  75  86              82.2
 10             50  55  70  65  55              59
 11             85  90  92  88  94              89.8
 12             90  92  88  95  95              92
 13             80  85  82  81  90              83.6
 14             82  80  86  80  90              83.6
 15             95  98  95  100  100            97.6
 16             86  95  88  85  90              88.8
 17             84  80  75  64  55              71.6
 18             86  75  84  85  80              82
 19             80  85  90  85  82              84.4
 20             70  75  65  78  70              71.6
 21             100  95  100  96  100           98.2
 22             85  85  80  90  92              86.4
 23             90  95  92  90  96              92.6
 24             70  60  75  68  50              64.6
 25             90  95  100  94  96             95
 26             88  85  70  80  74              79.4
 27             65  80  82  78  76              76.2
 28             85  88  84  86  90              86.6
 29             98  100  100  95  100           98.6
 30             75  80  85  85  90              83

READY
```

Observe: Again, the subscripted variables enable us to store and reference date in an array or matrix. We could have used G(I) and Z for grades 1 through 5 and the average **if** we wished to print out a line **before reading all data**. The program as is reads in all data **before processing.**

Exercises:

1. Write a program of your own choosing, using the DIM statement and a subscripted variable.

SUBROUTINES

It should be noted that we have already exceeded the "minimum" number of BASIC statements needed to solve all programming tasks. A minimum list might be the INPUT, LET, PRINT, and IF-THEN statements. The remainder of the BASIC statements are useful, however, in writing what we think of as "more efficient" or "shorter" programs. The GOSUB and RETURN commands are also unnecessary, but "useful" in certain programming tasks.

The use, if not the need, of these statements may be illustrated in a very simple program. We will use Newton's method of approximating square roots as our subroutine. (Suppose for the moment that the function SQR(X) is not available).

```
 10 REM USING NEWTON'S METHOD FOR APPROX SQUARE ROOTS AS
 20 REM A SUBROUTINE--PROGRAM CALCULATES 2ND,4TH,8TH, AND 16TH
 30 REM ROOTS OF POSITIVE NUMBERS
 40 PRINT "N","2ND ROOT","4TH ROOT","8TH ROOT","16TH ROOT"
 50 INPUT N
 60 LET R=N
 70 GOSUB 200
 80 LET R2=N
 90 GOSUB 200
100 LET R4=N
110 GOSUB 200
120 LET R8=N
130 GOSUB 200
140 LET R6=N
150 PRINT R,R2,R4,R8,R6
160 GO TO 50
200 LET G=1
210 LET H=0.5*(G+N/G)
220 IF ABS(H-G)<0.0001 THEN 250
230 LET G=H
240 GO TO 210
250 LET N=H
260 RETURN
1000 END
```

SUBROUTINE

TOLERANCE

```
RUN
N             2ND ROOT       4TH ROOT       8TH ROOT       16TH ROOT
? 256
 256           16             4              2              1.414214
? 0.1E17
 1.000000E+16  1.000000E+8    10000          100            10
? 81
 81            9              3              1.732051       1.316074
? 65536.5
 65536.5       256.001        16.00003       4.000004       2.000001
? 65536
 65536         256            16             4              2
?
```

Observe: Statements 200-260 make up the Newton's method subroutine. In this routine, an approximation H of the square root is compared with the previous approximation G. If the absolute value of their difference is less than a given tolerance (0.0001), then H is an acceptable approximation, and we return with the root H. This routine is used 4 times to obtain the 2nd, 4th, 8th, and 16th roots of N.

Observe: GOSUB 200 sends the control to the subroutine. RETURN sends control back to the **statement following** the GOSUB command.

Observe: Writing this routine as a **subroutine** avoided the necessity (apparently) of writing this set of commands 4 times.

Observe: The LIST command obtained the current program in storage. It has been modified and corrected many times before a satisfactory run was possible.

The above program illustrates the concept of the subroutine. Actually a simpler program would get the same results.

```
  10 PRINT "N","2ND ROOT","4TH ROOT","8TH ROOT","16TH ROOT"
  20 INPUT N
  30 FOR X=1 TO 5
  40 PRINT N,
  50 LET N=SQR(N)
  60 NEXT X
  70 GO TO 20
  80 END
RUN
N              2ND ROOT       4TH ROOT       8TH ROOT       16TH ROOT
? 256
 256            16             4              2              1.414214
? 0.1E17
 1.000000E+16   1.000000E+8    10000          100            10
? 81
 81             9              3              1.732051       1.316074
? 65536.5
 65536.5        256.001        16.00003       4.000004       2.000001
? 65536
 65536          256            16             4              2
?
```

Observe: The program above appears to obtain the same results. The FOR-NEXT loop generates the successive roots. Recognize that the operation of exponentiation and the SQR function may obtain slightly different results since these operations require built-in subroutines which may differ.

Format:

_____ GOSUB _____
(st. no.) (st. no.)

_____ RETURN
(st. no.)

Characteristics:

1. The statement number following GOSUB is the first statement of the subroutine.
2. RETURN is the **last statement** of the subroutine and returns control to the statement below (following) the GOSUB command.

Exercises:

1. Write a program of your own choosing, using the GOSUB and RETURN statements.
2. $y = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720}$ for values of x = 0, 0.2, 0.4, 0.6 . . . .2.8, 3.0.

   Compare these values with the values of the function y = cos x. (Print them side by side)
3. Write a program to solve the general system $Ax + By = C$ for x and y
   $Dx + Ey = F$
   for real values of the coefficients.
4. Write a program to obtain a list of primes.
5. Write a program that will list the prime factors of any natural number.
6. Write a program that will determine the probability of a number between 1 and 1000 being divisible by 7.
7. Write a program that will give the inverse of a given polynomial of the form $y = Ax^2 - Bx - c$.
8. Write a program that will determine the absolute value of a number without the use of the absolute value function.

IT'S A PLOT (Optional)

In all but the "smallest" BASIC packages, there is a graphing (plotting) function available. The TAB function provides us with carriage control (tabulation) so that we may place a character (or characters) in the print column of our choice. The TAB function is used only in the PRINT statement. Here are some examples of its use in graphing.

```
30 PRINT TAB(36) "*"
```

This causes an asterisk "*" to be printed in the 36th print column.

```
50 PRINT TAB(36+Y) "X"
```

This causes an "X" to be printed in the 36+Y column. If Y = -5, then "X" is printed in the 31st column.

```
35 PRINT TAB(30+0.5*Y) "*"
```

The 0.5 coefficient of Y causes a "shrinkage" of the graph horizontally so as to accommodate the plot.

```
40 PRINT TAB(30) "+" TAB(Y) "*"
```

This causes a "+" to be printed in the 30th column and "*" printed in the column determined by the value of Y.

The following program illustrates the plotting of the function $y = x^2 - 3x - 15$.

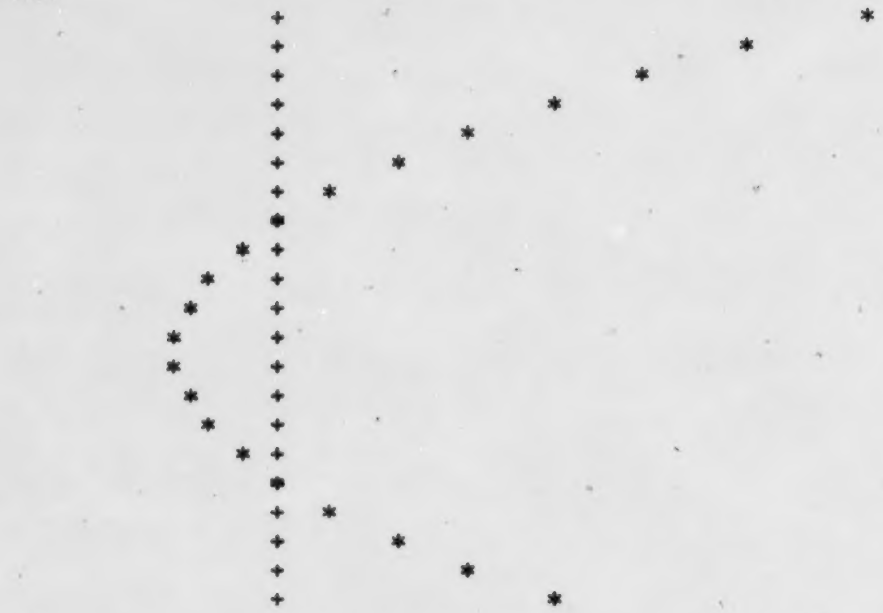
```
  30 FOR X=-10 TO 10
  40 LET Y=X↑2-3*X-15
  50 PRINT TAB(20+.3*Y) "*"
  60 NEXT X
  70 END
RUN
```

```
READY
```

The following addition to our program creates a "vertical" axis corresponding to y = 0.

```
50 PRINT TAB(20+.3*Y) "*" TAB(20) "+"
```

```
@TN←←RUN
```



```
READY
```

How about a little subprogram for a "horizontal" axis corresponding to x = 0.

```
32 IF X<>0 THEN 40
34 FOR I=5 TO 40
36 PRINT TAB(I) "+";
38 NEXT I
RUN
```

```
READY
```


62

Observe: The semicolon prevents a line feed while we print this axis.

The program as it now stands:

```
LIST

30 FOR X=-10 TO 10
32 IF X<>0 THEN 40
34 FOR I=5 TO 40
36 PRINT TAB(I) "+";
38 NEXT I
40 LET Y=X↑2-3*X-15
50 PRINT TAB(20+.3*Y) "*" TAB(25) "+"
60 NEXT X
70 END

READY
```

We could, of course, dress the program up with coordinates and other novelties. We could also print out a list of corresponding values of x and y.

In case you were wondering, since there are some 72 distinct print columns, the TAB function truncates decimal values calculated in its argument (within the parentheses).

Exercise:

1. Write a program of your own choosing which employs the TAB function.

APPENDIX I

Control Commands, Error Messages, and Editing

CONTROL COMMANDS

There is considerable communication that must take place between the operator and the computer that is not part of the program itself. The RUN command, given by the operator, signals the computer to begin processing the stored program.

```
  10 PRINT "HELLO, WORLD!"
  20 END
RUN
HELLO, WORLD!

READY
```

PROGRAM

Entered by Operator

Output

Response by Computer

READY is the response **by the computer** which signals the operator that processing of the program is finished or that a new program may be entered.

There will be a command like LIST, given by the operator by teletypewriter, that will cause the computer to print out the program in memory. This is particularly useful after a number of changes have been made in a program.

```
10 PRINT "PEOPLE ARE BEAUTIFUL!"
20 END
RUN
PEOPLE ARE BEAUTIFUL!

READY

LIST
  10 PRINT "PEOPLE ARE BEAUTIFUL!"
  20 END

READY
```

Command entered by Operator

Computer Output—a "list" of the stored Program

These three commands are quite universal among BASIC software packages.

Once a program is entered by the operator, it will remain until cleared from memory. Thus, it is essential that the memory be cleared before new programs are entered. Commands usually given for this purpose are NEW or SCRATCH.

```
10 PRINT "PEOPLE ARE BEAUTIFUL!"
20 END
RUN
PEOPLE ARE BEAUTIFUL!

READY

NEW
```

← Entered by Operator — clears the program from Memory

```
READY

LIST
```

← No program in Core now!

```
READY
```

Observe: LIST was typed to see what was in memory and it can be seen that the program was indeed cleared from memory.

Another direct command will interrupt the run of a program during its processing. Such a command is useful when it is seen that the program is in an endless loop or that the desired results are not being obtained. This interrupt may be brought about by striking the "S" (for stop) key in some systems or by depressing the CTRL and C keys in others

```
 10 REM J. STOUDT 8/50/70
 20 REM COUNTING EXERCISE
 30 PRINT X
 40 LET X=X+1
 50 GO TO 30
 60 END
RUN
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
```

Observe: The preceding program was interrupted by depressing the CTRL and C keys.

```
 10 REM J. STOUDT 8/50/70 --COUNTING EXERCISE--DIFFERENT SYSTEM
 20 PRINT X
 30 LET X=X+1
 40 GO TO 20
 50 END
RUN
RUN
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11

STOP

READY
```

Observe: The system used for this problem requires that S be keyed in order to stop processing. STOP is then printed.

There are other commands, of course, for punching and reading tape, for editing, and other purposes. Since these commands do vary from system to system, the reader should consult the operating manual of the system in use.

ERROR MESSAGES

Most BASIC systems are supported by rather literal error messages. In the program run below, a simple English statement carries the message.

```
 10 LET X=12.589
 20 LET Y=.12589E+02
 30 LET Z=12589E-02
 40 LET I=X*Y*Z
 50 PRINT "THE PRODUCT OF X,Y,Z IS"; I
RUN

ERROR  33


READY

60 END
RUN
THE PRODUCT OF X,Y,Z IS 19951.42

READY
```

Observe: After RUN was given by the operator, the computer typed an error message and then READY An END statement was added by the operator and the program was processed.

Often the error message will indicate by number the **type** of error and then indicate the **line** in which the error occurs.

```
SCR

READY

  50 LET D=2
  60 LET F=5
  70 PRINT D*F, D^F, 4+D/F, D/(F-5), F*(D-2)
  80 END
RUN
 10              32              4.4
ERROR   3 IN   70

READY
```

Observe: "Error 3 in 70" indicates that "division by zero is requested" in statement 70. The error code numbers will vary from system to system.

```
  10 LET D=2
  20 LET F=5
  30 PINT D*F, D/F
  40 END
RUN

ERROR  48 IN   30

READY
```

Observe: Error 48 is "unrecognized command."

```
  10 LET X=5
  20 PRINT (2*X+5)*X+6)*X+4
  30 END
RUN
 81
ERROR  32 IN   20

READY
```

Observe: Error 32 in this system indicates "syntax in print." The computer did not compute the complete expression and balked because of a missing parenthesis.

PROGRAM MODIFICATION

It is certainly common that mistakes are made upon entering programs in the computer. There are several ways to correct errors and modify statements.

If a character or two has been mistyped, the operator may immediately strike the RUBOUT key once for each incorrect character.

```
1Ø PRINT "ECOLOGY IS A MATTER OF CONXZ++CERN TO EVERYONE."
2Ø END
RUN

ECOLOGY IS A MATTER OF CONCERN TO EVERYONE.

READY
```

If a line is to be modified or corrected, the operator may do so by simply typing the statement over (until it is correct).

```
  1Ø LET X=5
  2Ø PRINT (2*X+5)*X+6)*X+4
  3Ø END
RUN
 81
ERROR  32 IN   2Ø

READY

2Ø PRINT ((2*X+5)*X+6)*X+4
RUN
 4Ø9

READY
```

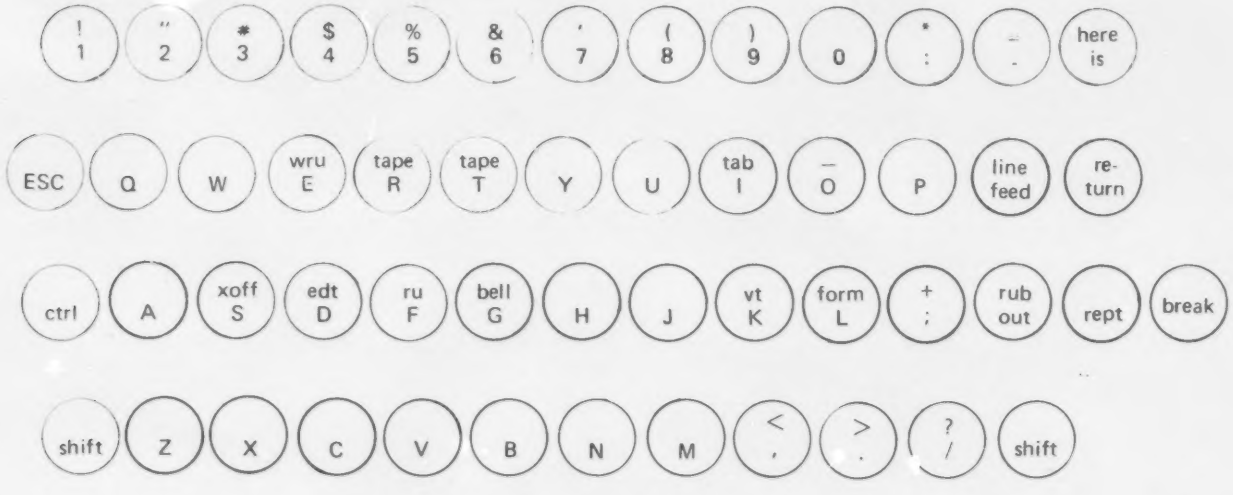
Observe: Statement 20 is simply typed again to include the missing parenthesis.

The program on the following page demonstrates modification by typing a statement over and then the use of LIST to see that the statement finally was entered correctly.

```
25 PRINT "N", "N SQUARED", "N CUBED" , "N
25 PRINT B
25 PRINT "N", "N SQUARED", "N CUBED", "N↑4","N↑5"
LIST

  10 REM J.  DOE 7/32/70
  20 REM POWERS OF ODD NUMBERS WITH 5
  25 PRINT "N","N SQUARED", "N CUBED", "N↑4","N↑5"
  30 LET N=5
  40 PRINT N,N↑2,N↑3,N↑4,N↑5
  50 LET N=N+2
  60 GO TO 40
  70 END
RUN
N              N SQUARED      N CUBED        N↑4            N↑5
 5              25             125            625            3125
 7              49             343            2401           16807
 9              81             729            6561           59049
 11             121            1331           14641          161051
 13             169            2197           28561          371293
STOP

READY
```

The command EDIT 40 will also set up line number 40 for modification. Most systems also provide commands that order the computer to print a line up to a given "search" character which may then be corrected. Again, the operator should consult the operating guide for particulars.

Until the reader has advanced beyond this manual, the retyping of lines and the use of the RUBOUT key should be adequate means of program modification.



TELETYPE CORPORATION
33 Teletypewriter ASR Keyboard

APPENDIX II

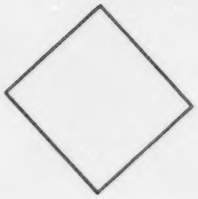
Flow Charting

Flow charting is a visual means by which the programmer plans the solution of complex problems. The programmer diagrams the step-by-step process through the use of arrows and other symbols. The flow chart can be a useful aid to the problem solver; to help him/her over the tough logical sections of his/her program. Once the flow chart has been prepared, the program can easily be written.

There are four basic flow-chart symbols in common use. They are:

1.

This symbol is used to denote a **process.** We might use this symbol to indicate an addition like C = A + B, subtraction, or "replacements," like A = A + 1.

2. 

This symbol is used to denote a **decision** that must be made.

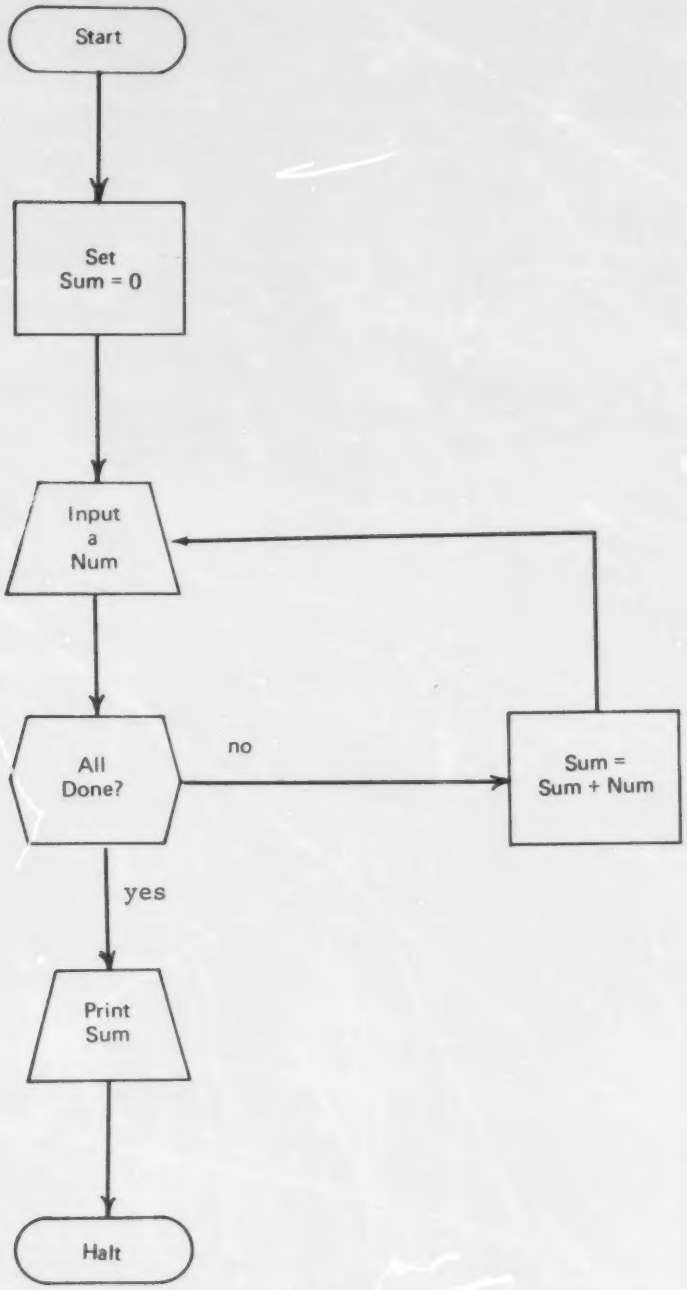
3.

This symbol denotes the **input** or **output** of data.

4.

This symbol denotes the **start** or **halt** of a program.

The following illustrates the flow chart of a simple problem to find the sum of a series of numbers:



Once a flow chart has been prepared, the program "coding" is usually not difficult.

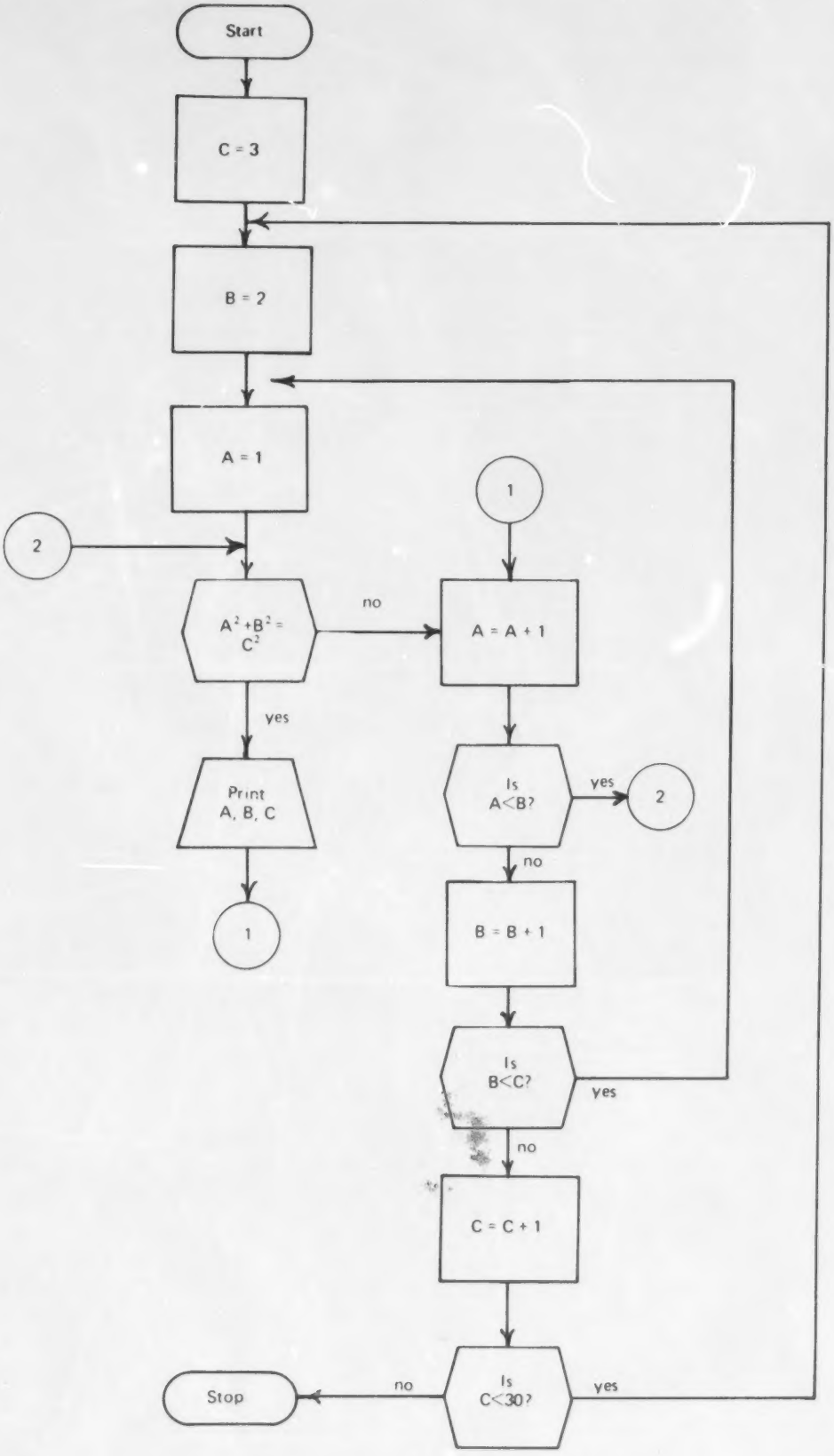
```
LIST

  10   LET S=0
  20   INPUT N
  30   IF N= -999 THEN 60
  40   LET S=S+N
  50   GO TO 20
  60   PRINT S
  70   END

READY

RUN
? 5
? 6
? 17
? 25
? -999
  53

READY
```

Observe: Statement 30 tests the input number. The number -999 signals the end of input data and causes a branch to the PRINT statement.

There is no universal vocabulary or symbolism in use by programmers, but differences from flow chart to flow chart are not great. The flow chart will ordinarily contain abbreviations and conversational words and phrases like "IS NUM -999?" The flow chart, in a sense, is just one step away from the abstraction found in the programming language. In the problem formulation, we are talking about "numbers," in the flow chart, "NUM," and in the program we used the variable name "N." Likewise, the phrase "IS NUM -999?" is perhaps more natural than the IF-THEN statement in the program.

It must be said that some programmers rely heavily on the flow chart and others rarely use them. But, the flow chart does play a big role in the documentation of a program, and many firms would consider it a necessity. If one is asked to acquaint oneself with a program, the flow chart might be the best place to begin.

On the following page is a flow chart for the generation of pythagorean tripies.


Start
C = 3
B = 2
A = 1
2
A² + B² = C²
no
yes
Print A, B, C
1
1
A = A + 1
Is A<B?
yes
2
no
B = B + 1
Is B<C?
yes
no
C = C + 1
Is C<30?
yes
no
Stop

Observe: The "IS C<30?" decision provides a stop when we have generated the requested triples. There are eight sets of triples for C<30.

The coding of this program is quite easy now because the flow chart accomplished the greater task of providing an algorithm (assuming it will work) for generating pythagorean triples.

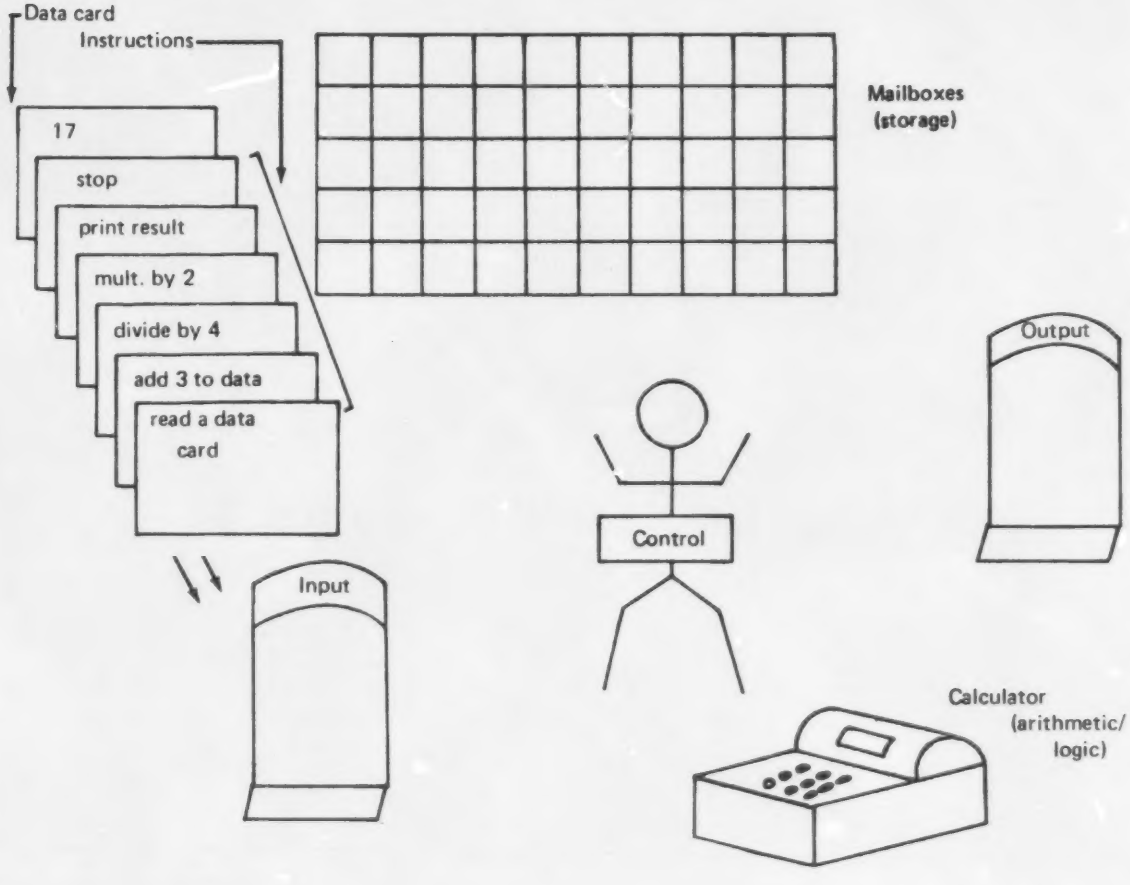
APPENDIX III

(A Useful Analogy)

A number of authors have used the "mailbox analogy" to illustrate the operation of the digital computer. We will borrow the analogy. It seems to serve instructional purposes quite well and might be quite appropriate for the secondary school classroom. Recognize that certain liberties are taken in order to simplify the operation.

This analogy makes use of card input as opposed to teletypewriter input. The instructions (statements) are not written in BASIC, FORTRAN, or any other recognized programming language, but the principles stressed here are applicable to all digital computing systems.

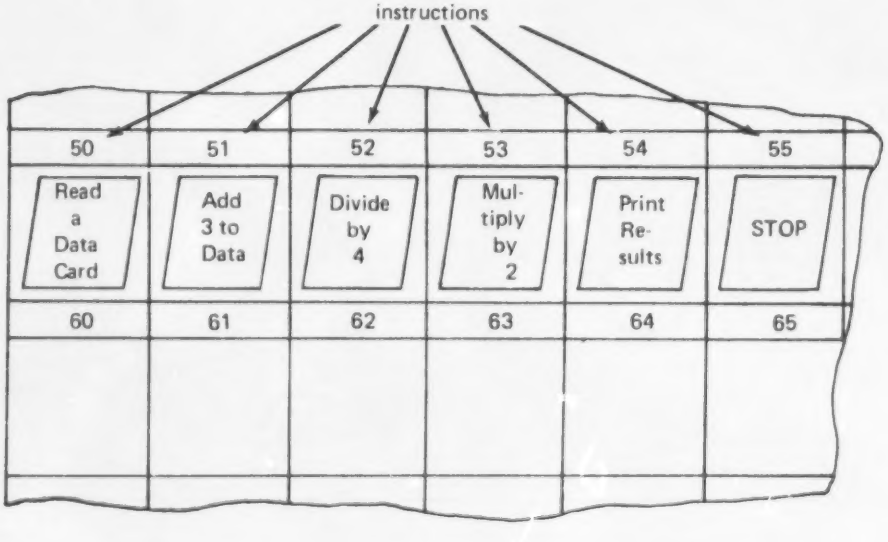
Suppose we deliver to a letter carier (control unit) a stack of cards: 1) some of which have a single instruction written on them, and 2) some of which have data written on them. (Only one piece of data is involved in this illustration).



a) First, the letter carrier (control unit) stores each instruction in mailboxes (storage).

1	2	3	4	5	6	7	8
11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28

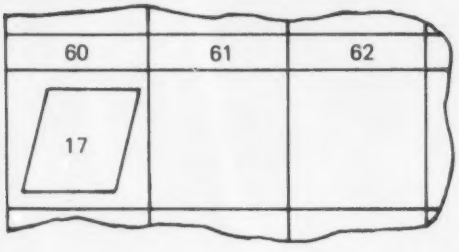
The mailboxes must be numbered (addressed) for reference. The letter carrier (control unit) will place the instructions in preset locations and in sequential order. In our illustration, let us assume they are placed in addresses 50-55 (six instructions - six storage units). The data card is not yet stored.



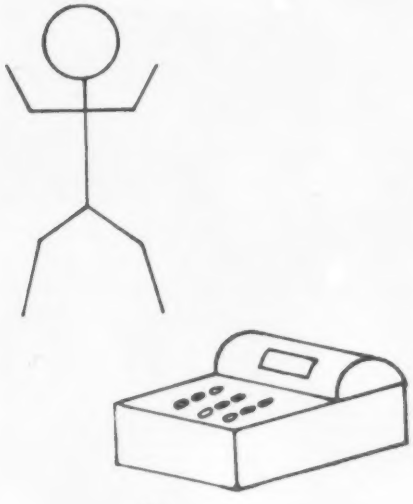
Now, with the instructions in storage, our machine continues.

b) The control unit goes to the address of the first instruction. That instruction tells him/her to read a data card. In so doing, he/she places the data in a storage location. (Let's use address 60).

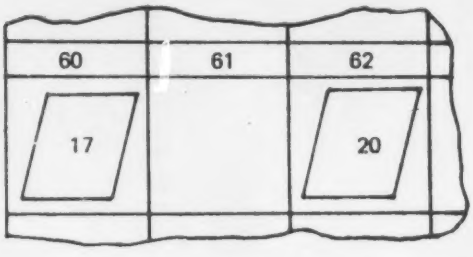
Note: The read instruction is not destroyed, and remains in location 50.



c) Control then goes to address 51 which is the next instruction. That instruction directs our machine to add 3 to the data quantity. The control knows that the data is address 60. He/she performs the addition with the calculator (arithmetic-logic unit).

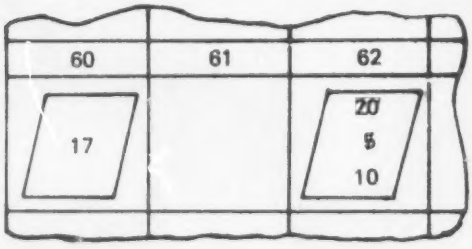


The sum of 17 + 3 = 20 must be retained for the moment. Let's have it stored in address 62 (an arbitrary choice).

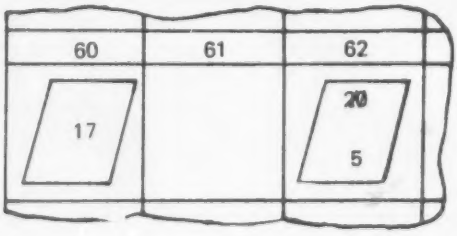


Note: Again, neither the instruction (address 51) nor the data (address 60) is destroyed. Actually, we might have chosen to store our sum (now in 62) in address 60 since we no longer need the original data (17). An instruction or piece of data is destroyed only if we place something else in that address.

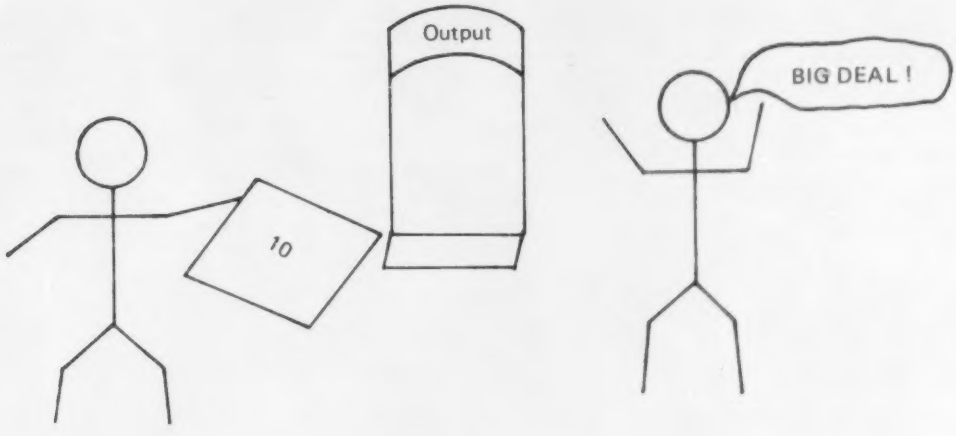
d) Control then goes to the next instruction (address 52) which directs the machine to divide the sum (address 62) by 4. This quotient 20 ÷ 4 = 5 again must be retained for further use, but this time let's just put it back in address 62. (We do not need the 20 any longer!)



e) Control goes to the next instruction address (53) which directs the machine to multiply the dividend by 2. Again the calculator (arithmetic-logic) gives us the result: 5 x 2 = 10. We use our "smarts" and place this quotient back into "good ole" address 62.



f) (The end is in sight!) Control goes to the next instruction (address 54) which directs the machine to print the **result**. The number 10 (a carbon copy, actually) is obtained from address 62 and printed on an output sheet or card.



THAT'S IT!!

Let us review some very important and rather universal characteristics of the digital computer as illustrated in our analogy.

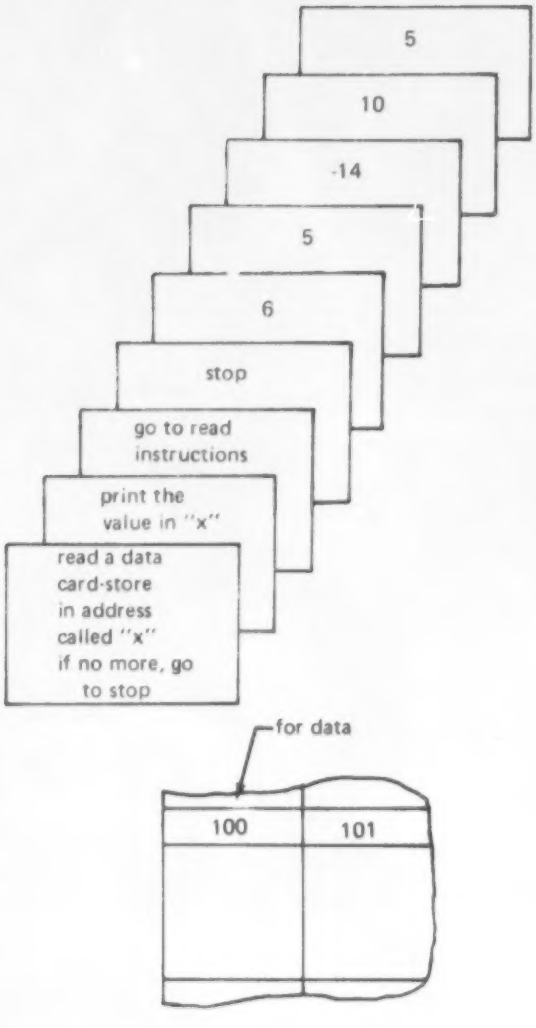
1) Instructions are "loaded" into storage sequentially in preset addresses. You may never learn the actual addresses in a particular machine, but "it" knows! The machine "steps" through the instructions sequentially unless told to do otherwise.

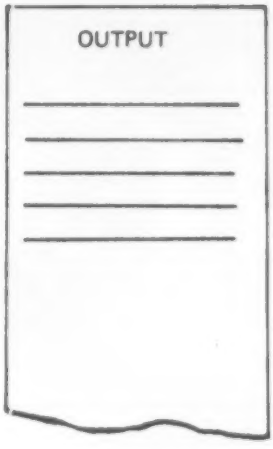
2) A "read" instruction causes the computer to place a piece of data in storage.

3) When instructions are performed or when arithmetic is performed on data, those instructions and data are not destroyed. Destruction of instructions or data occurs **only** when another instruction or piece of data is placed in that same address. Sometimes we destroy data or change instructions intentionally for a variety of reasons. A common mistake in programming is the unintentional destruction of data.

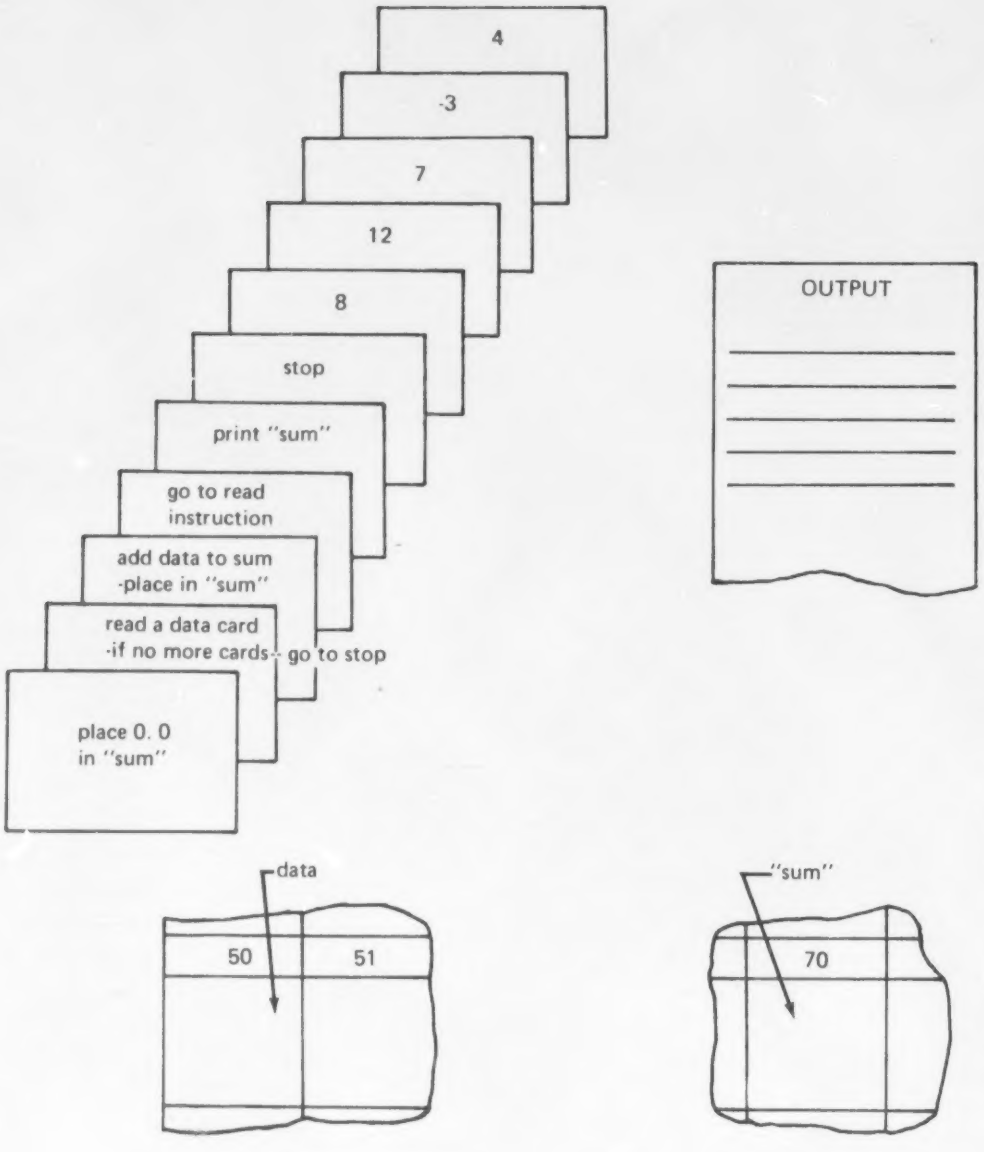
Exercises:

1) Consider the following program. Let us assume the instructions are already "loaded". "Walk" through the instructions, recording in pencil (and making appropriate erasures) data as they are placed, moved, or modified in storage. Also print out results when told to do so. Address 100 will be used to store input data.





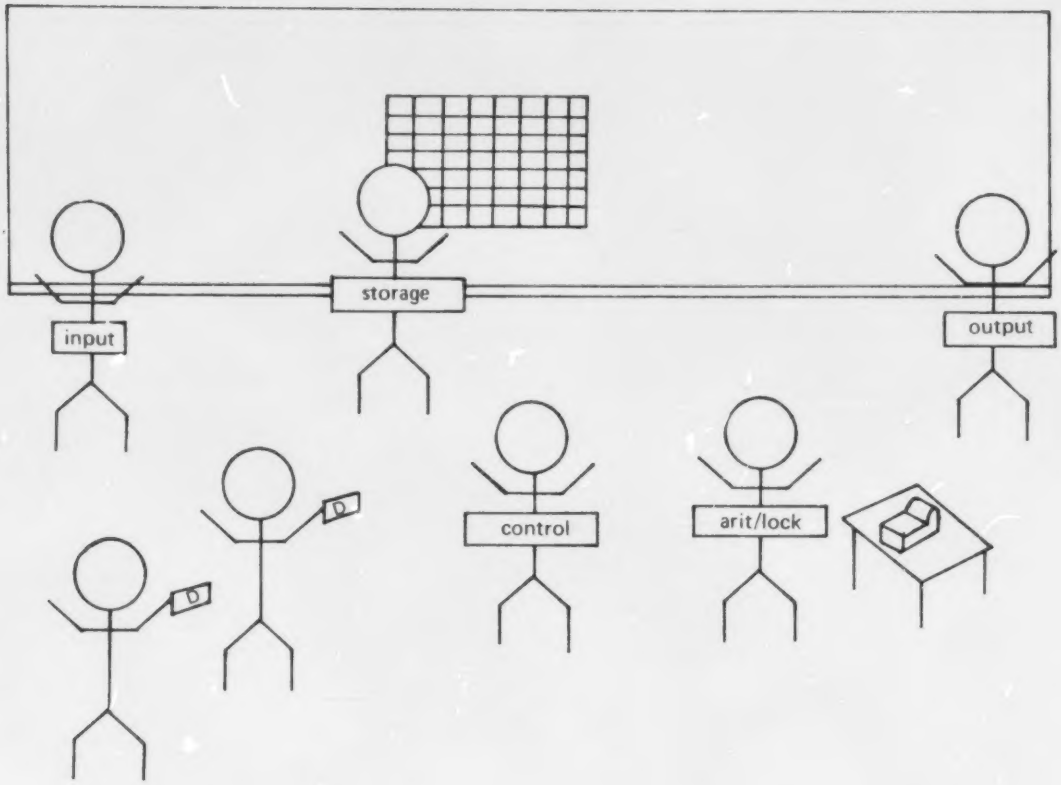
2) Consider the following program and assume that the instructions are already loaded into storage. Notice that location 70 will be used for our accumulated "sum" and data will be read into location 50. Use pencil (and eraser) to record the sum as it is accumulated in location 70. "Walk" through the program.



Activities:

1) Younger students might profit from the following activity. Set up the room as a computer, using students to fill the roles of the input device (he/she "reads" and stores data from cards); the storage unit (he/she uses the chalkboard to record data in addressed boxes); the arithmetic-logic unit (he/she carries out calculations); the output device (he/she writes output on the board or other medium); and the control unit (he/she directs the whole operation).

— see illustration next page —



Other students can hold the data cards until read by the input device. The teacher may want to act as control in the beginning to keep the computer functioning smoothly.

Here are two programs that might be used:

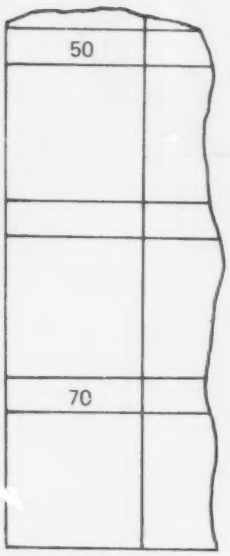
Note: It is important to recognize that in these problems we refer to the somewhat temporary storage locations for data as "feet," "inch," "cube," etc., rather than the numeric addresses. This is a step toward symbolic programming.

a) **Conversion Program**

```
READ, STORE IN "FEET" - IF NO MORE CARDS, GO TO STOP
MULTIPLY "FEET" BY 12, STORE IN "INCHES"
MULTIPLY "INCHES" BY 2.54, STORE IN "CM"
PRINT "FEET," "INCHES," "CM"
GO TO READ STATEMENT
STOP
3
4                    data
5
6
```

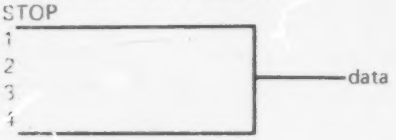
Note: Place the instructions in addresses 101-106, and assign "Feet" to 50, "Inches" to 60, and "CM" to 70.

– see illustration next page –



b) **Squaring, and Cubing Numbers**

READ, STORE IN "NUMBER" - IF NO MORE CARDS, GO TO STOP
MULTIPLY "NUMBER" BY "NUMBER," STORE IN "SQUARE"
MULTIPLY "SQUARE" BY "NUMBER," STORE IN "CUBE"
WRITE "NUMBER," "SQUARE," and "CUBE"
GO TO READ STATEMENT
STOP



Note. Once again we will need 6 (sequential) storage locations for the Instructions and we must have three storage locations (choose your own) for "NUMBER," "SQUARE," and "CUBE."

Answers to Selected Exercises

Page 5:

1. 9
2. 45
3. 6
4. 27
5. -3
6. 64
7. 4.41421
8. 16
9. 45
10. 29.5

Pages 7 & 8:

1. 6.75 1.2 Ø.Ø27
   6.75 1.2 Ø.Ø27
   READY

2. THE EXPRESSION 7*(8-2)/5 EQUALS
   8.4
   READY

3. SUM DIFFERENCE PRODUCT QUOTIENT EXPONENTIAL
   188.955 62.Ø4 2.4 70 7
   READY

4. THE COORDINATES ("3+4-5 ", "7.Ø*2-9 ") DESIGNATE THE POINT P1.
   READY

Page 14:

1. 7 1 12 1.333333 64

2. 7 1 12 1.333333 64

5. 2 6
   2 8

Page 26 & 27:

1. No output

2. 55

3. 1
   1
   2
   3
   5
   5
   8

Page 44:

1. -5
   -3
   -1
   1
   3
   5

   -5
   -2
   1
   4

   -5
   -1
   3